AFRL-RI-RS-TR-2018-039

FOUNDATIONS OF SEQUENTIAL LEARNING

DUKE UNIVERSITY

FEBRUARY 2018

FINAL TECHNICAL REPORT

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)* FEB 2018	**2. REPORT TYPE** FINAL TECHNICAL REPORT	**3. DATES COVERED** *(From - To)* APR 2016 – AUG 2017

4. TITLE AND SUBTITLE FOUNDATIONS OF SEQUENTIAL LEARNING	**5a. CONTRACT NUMBER** FA8750-16-2-0173
	5b. GRANT NUMBER N/A
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Ronald Parr, Kamesh Munagala, Cynthia Rudin	**5d. PROJECT NUMBER** FOSL
	5e. TASK NUMBER 20
	5f. WORK UNIT NUMBER 16
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Duke University Office of Research Administration 2200 W. Main St., Ste. 710 Durham NC 27708-4677	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RITA 525 Brooks Road Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/RI
	11. SPONSOR/MONITOR'S REPORT NUMBER AFRL-RI-RS-TR-2018-039

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report summarizes the research done under FA8750-16-2-0173. This research advanced understanding of bandit algorithms and exploration in Markov Decision Processes (MDPs). New algorithms and theory were proposed for bandits with periodic payoff multipliers and arms with costs. Exploration and transfer learning algorithms were evaluated for MDPs.

15. SUBJECT TERMS
Machine learning, bandit algorithms, reinforcement learning

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** CHRISTOPHER J. FLYNN
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	95	**19b. TELEPHONE NUMBER** *(Include area code)*

Contents

1. Summary 1

2. Introduction 1

2.1 Multi-armed Bandits 1

2.2 Budgeted Multi-Armed Bandits 1

2.3 PAC-optimal Transfer Learning in MDPs 2

3. Methods, Assumptions and Procedures 2

3.1 Bandit Problems 2

3.1.1 ε-greedy 2

3.1.2 UCB 2

3.1.3 Thompson Sampling 2

3.2 PAC-optimal exploration in MDPs, and transfer in MDPs 2

4. Results and Discussion 3

4.1 Bandits with Fluctuations 3

4.2 Budgeted Bandits 3

4.3 Transfer in MDPs 3

5. Conclusions 4

Appendix A Regulating Greed Over Time 5

Appendix B Multi-armed Bandit Problem 32

Appendix C Sample efficient PAC Reinforcement Learning 52

Appendix D Pac Intra-task Knowledge Transfer in MDPs 67

Appendix E Transfer with GAN in Policy Gradient Reinforcement Learning 75

Appendix F PAC Reinforcement Learning Knowledge Transfer via Manifold Alignment 81

6. List of Acronyms 91

1. Summary

The investigators developed and explored new algorithms for multi-armed bandits with periodic fluctuations, multi-armed bandits with budgets, and for transfer learning in Markov Decision Processes (MDPs). In the periodic fluctuations case, they developed new theory and promising experimental results. In the budgeted case, there are promising experimental results but no new theory. In the MDP case, existing theory was validated on small problems but did not generalize well to larger problems due to computational limitations.

2. Introduction

This project involved three thrusts:

1. New algorithms for multi-armed bandits (Rudin).
2. New algorithms for multi-armed bandits in the presence of costs and a budget (Munagala).
3. Evaluation of transfer learning algorithms for Probably Approximately Correct (PAC)-optimal learning in MDPs (Parr).

The three investigators explored each of these questions and produced some new algorithms and promising experimental results, as described in the subsequent sections and attached documents.

2.1 Multi-armed Bandits

Multi-armed bandits abstract a basic learning question in which the learned must estimate the payoff or reward that results from making different choices (abstracted as arms on different slot machines). The payoffs are stochastic, so a single pull is not sufficient to estimate the payoff. The learned must use a strategy that balances exploration (learning new information by pulling arms that might not have had the highest payoff so far) with exploitation (pulling the arm that is currently estimated to have the highest payoff).

Rudin's team observed that payoffs may often have periodic fluctuations that can be exploited, a case that was not addressed by existing literature. She developed new algorithms and results, summarized in the next section, and discussed in detail in Appendix A1.

2.2 Budgeted Multi-Armed Bandits

Budgeted Multi-Armed bandits are a generalized over the standard multi-armed bandit problem to the case where arms have an associated cost and there is a fixed budget for exploration. This is, arguably, a more realistic scenario in which there is a variable cost associated with exploring various options and this cost must be taken into account.

Munagala's team developed a new algorithm for the budgeted case and were able to show that this algorithm performed better than the natural alternatives in simulations. This is summarized in the next section, and greater detail is provided Appendix A2.

2.3 PAC-optimal Transfer Learning in MDPs

PAC-optimal learning in MDPs involves learning how to perform near optimally in an MDP while making a bounded number of steps that are significantly suboptimal. Existing work in this area had very limited ability to transfer knowledge from one problem to a related problem.

In unpublished previous work (see appendix A3) Parr and coauthors developed new algorithms for transfer learning that included the use of a *transfer function* to transfer knowledge between MDPs. This allowed new PAC-optimality bounds. However, it was not clear if the theory would be useful in practice.

Parr's team implemented and tested the above approach. The results are summarized in the next section and described in detail in Appendix A4. In addition, they explored the use of various manifold alignment and Generative Adversarial Network (GAN) techniques to achieve practical transfer learning results.

3. Methods, Assumptions and Procedures

3.1 Bandit Problems

Standard approaches to bandit problems involve some attempt to balance exploration with exploitation. The techniques explored in his project involve generalizations of the three main approaches described below:

3.1.1 ε-greedy

The ε -greedy approach picks the arm that looks best with probability 1- ε, and picks a random arm with probability ε. The parameter may be chosen adaptively.

3.1.2 UCB

The Upper Confidence Bound (UCB) approach estimates the payoff for each arm within a confidence interval and adds a decaying (with experience) exploration bonus to ensure that even suboptimal arms are continually tested - albeit decreasingly with time.

3.1.3 Thompson Sampling

Thompson sampling is a Bayesian approach that maintains a distribution over payoffs. Models are sampled from this distribution and the distribution is updated based upon the results. As with UCB, this ensure that arms will be tried with some positive probability throughout learning.

3.2 PAC-optimal exploration in MDPs, and transfer in MDPs

Similar to the UCB algorithm mentioned above, most PAC-optimal algorithms for MDPs involve some form of exploration bonus. Exploration in continuous state spaces typically involves some form of state aggregation so that similar states are clustered together to create a piecewise-constant value function. This was done in Parr's previous work.

Transfer learning in MDPs (and reinforcement learning) can take many forms, though there is very little work on sample complexity for this case. Parr's work involves a user-provided transfer function that shows how samples from one MDP can be adapted to another MDP.

4. Results and Discussion

4.1 Bandits with Fluctuations

Rudin’s team developed several new algorithms for this case. She introduced a *greed* parameter that could be regulated over time. The regulation of greed is done in response to a known, external signal that acts as a payoff multiplier. For example, a retailer may evaluation various strategies for reaching consumers at different times of the year, but during the December holiday season, the payoffs for various strategies could be multiplied because of increased consumer spending. The retailer may have a good estimate of the multiplier but not the base payoffs.

Rudin considered three variations on existing work:

1. A variable arm pool approach that limits or expands the number of arms considered based upon the payoff multiplier.
2. Variations on the -greedy approach that takes the multiplier into account.
3. Variations on the UCB algorithm that take the multiplier into account.

In each of the above cases, Rudin proved regret bounds for the modified algorithms. In addition, experiment results showed that these algorithms performed favorably in comparison to the standard algorithms in presence of periodic payoff multipliers.

4.2 Budgeted Bandits

Munagala’s team modified the Thompson sampling approach for multi-armed bandits to the budgeted cost case. Experimentally, this approach was shown to be superior to existing algorithms for the budgeted case, though he was not able to obtain any positive theoretical results to support the experimental observations.

In addition, Munagala’s team showed theoretically that no algorithm could be expected to have reasonable performance against an adversary that chose arms costs in an adversarial manner. Allowing an adversary to set costs implies a very strong adversary, to a weaker model that allowed costs to change slowly was considered and a new algorithm was proposed, though it did not perform consistently better than other approaches.

Munagala also considered a generalization of the Thompson sampling algorithm to contextual bandits, though the experimental results were not promising. In the final phase of the project (not detailed in the appendices), the team considered a contextual bandit case where there is a global context that that multiplies the payoffs (similar to the case considered by Rudin). In this case, the modified version of Thompson sampling did provide some advantage over existing algorithms.

4.3 Transfer in MDPs

Parr’s team conducted experiments to evaluate the benefit of using a transfer function to improve transfer learning MDPs. The transfer function describes how samples from one MDP can be transformed to act like samples in another MDP (or a related part of the same MDP). For example, in a single MDP with that exhibits some sort of symmetry, samples from one quadrant of the state space could be transformed to act like samples from another part of the state space by flipping the signs of the state variables, effectively doubling the number of reduces and reducing the sample complexity by half.

Experimental results for a simple problem like the classic inverted pendulum benchmark showed that the expected reduction in the number of samples required did indeed occur. Unfortunately, it proved difficult to find interesting examples that satisfied the assumptions of the theory and that also had easily described transfer functions that provided significant reduction in sample complexity.

Parr's team also considered other approaches to transfer such as the use of GANs. Initial results, detailed in Appendices A5 and A6, were somewhat promising, but like many GAN methods, it was somewhat unstable. Prospects for extending this approach to more challenging problems are unclear. It is possible that improved GAN techniques that have been developed recently could help.

5. Conclusions

In the case of multi-armed bandits, a major finding of this project is that periodic noise multipliers can be exploited. This is supported by theoretical results and experimental results. For arms with budgeted costs, variations on Thompson sampling look promising, though theoretical results supporting the experimental results are still lacking.

For transfer learning in MDPs, the experimentation done supports the existing theory, but existing algorithms with theoretical guarantees still do not scale well to large problems. Other techniques, such as transfer through GANs, show some promise, but further work is required to improve the stability of such approaches and demonstrate that they can work on larger and more challenging problems.

APPENDIX A Regulating Greed Over Time

Regulating Greed Over Time

STEFANO TRACÀ AND CYNTHIA RUDIN

Abstract

In retail, there are predictable yet dramatic time-dependent patterns in customer behavior, such as periodic changes in the number of visitors, or increases in visitors just before major holidays (e.g., Christmas). The current paradigm of multi-armed bandit analysis does not take these known patterns into account, which means that despite the firm theoretical foundation of these methods, they are fundamentally flawed when it comes to real applications. This work provides a remedy that takes the time-dependent patterns into account, and we show how this remedy is implemented in the UCB and ε-greedy methods. In the corrected methods, exploitation (greed) is regulated over time, so that more exploitation occurs during higher reward periods, and more exploration occurs in periods of low reward. In order to understand why regret is reduced with the corrected methods, we present a set of bounds that provide insight into why we would want to exploit during periods of high reward, and discuss the impact on regret. Our proposed methods have excellent performance in experiments, and were inspired by a high-scoring entry in the Exploration and Exploitation 3 contest using data from Yahoo! Front Page. That entry heavily used time-series methods to regulate greed over time, which was substantially more effective than other contextual bandit methods.

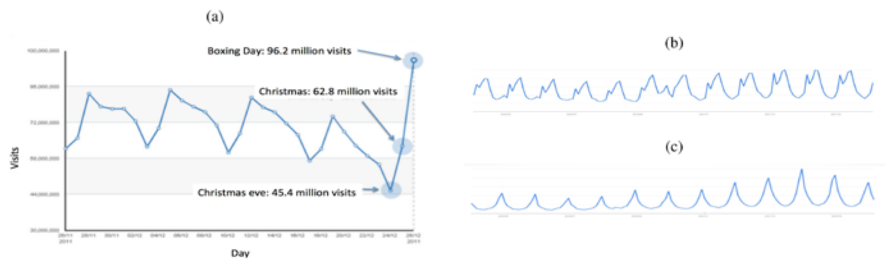
Keywords: Multi-armed bandit, exploration-exploitation trade-off, time series, retail management, marketing, online applications, regret bounds.

1 Introduction

Consider the classic pricing problem faced by retailers, where the price of a new product on a given day is chosen to maximize the expected profit. The optimal price is learned asymptotically through a mix of exploring various pricing choices and exploiting those known to yield higher profits, potentially through the use of a multi-armed bandit (MAB). We assume the retailer knows the daily trend of the number of customers over time that visit the store. This information can be leveraged in order produce a better exploration/exploitation scheme. For instance, if we know that many customers will come to the store on the week before Christmas, we would not want to explore new prices on those days. We might even stop exploring all together. Our setting violates the classic assumptions of random rewards with a static probability distribution that is typically considered in multi-armed bandits. This is because rewards are correlated through the trends in customer behavior. If one uses a standard MAB algorithm in the case where trends are dramatic, the result could be arbitrarily bad. A simple example is the case where the number of customers at the store will have a predictably large spike on a given day (e.g. for boxing day in England, shown in Figure 0a), where the classic MAB algorithm could choose a poor price on that particular day for the purpose of exploration.

For retailers, there are almost always clear trends in customer arrivals, and they are often periodic or otherwise predictable. Some examples are in Figure 0b and 0c. These dramatic trends might have a substantial impact on which policy we would use to price products. The main contributions of this work are (i) A new framework that illustrates when it is beneficial to stop exploration sometimes to favor exploitation. (ii) Novel algorithms that show how to adapt existing policies to regulate greed

Figure 1: (a) English users shopping online. Source: ispreview.co.uk. (b) Google searches for “strawberries". Source: Google trends. (c) Google searches for “scarf". Source: Google trends.



over time. These are: Algorithm 2: ε-greedy algorithm with regulating threshold (Section 3.3); Algorithm 3: soft ε-greedy algorithm (Section 3.4); Algorithm 4: UCB algorithm with regulating threshold (Section 3.5); and Algorithm 5: soft UCB algorithm (Section 3.6). (iii) Theoretical regret bounds for the above algorithms. (iv) Numerical comparisons (in Appendix 4). We compare to "smarter" versions of the classic ε-greedy algorithm (Algorithm 7) and UCB algorithm (Algorithm 6). The standard algorithms incorrectly estimate the mean rewards of the arms. The "smarter" versions fix that issue, and thus are a reasonable baseline to compare with. The "smarter" algorithms do not regulate greed over time however, and are not comparable in performance to the algorithms that do this regulation.

In our setting, the behavioral information about customers is distilled so that it takes the form of a *reward multiplier* $G(t)$, where we assume $G(t)$ is known or can be well-estimated before the decision is made at time t. $G(t)$ should be thought of as the number of customers in the store on day t. If $G(t)$ is not known but could be well-approximated, the regret bounds weaken accordingly. The new algorithms, that take advantage of knowing $G(t)$, are not a simple extension of the ε-greedy algorithm and the UCB algorithm. They anticipate the number of customers and choose how much exploration to allow at that timestep.

As a result of the reward multiplier function, theoretical regret bound analysis of the multi-armed bandit problem becomes more complicated, because now the distribution of rewards depends explicitly on time. We not only care *how many* times each suboptimal arm is played, but exactly *when* they are played. For instance, if suboptimal arms are played only when the reward multiplier is low, intuitively it should not hurt the overall regret.

2 Related Work

The setup of this work differs from other works considering time-dependent multi-armed bandit problems – we do not assume the mean rewards of the arms exhibit random changes over time, and we assume that the reward multiplier is known in advance. Other works consider different scenarios where reward distributions can change over time, but in a way that is not known in advance. For these settings, the algorithm needs to compensate for changes in the reward distribution after the change, rather than altering their strategy in advance of the change. Along these lines, [Liu et al., 2013] consider a problem where each arm transitions in an unknown Markovian way to a different reward state when it is played, and evolves according to an unknown random process when it is not played. [Garivier and Moulines, 2008] presented an analysis of a discounted version of the UCB and a sliding window version of the UCB, where the distribution of rewards can have abrupt changes and stays stationary in between. [Besbes et al., 2014] considers the case where the mean rewards for each arm can change, where the variation of that change is bounded. [Slivkins and Upfal, 2007] consider an extreme case where the rewards exhibit Brownian motion, leading to regret bounds that scale differently than typical bounds (linear in T rather than logarithmic). One of the works that is relevant to ours is that of [Chakrabarti et al., 2009] who consider "mortal bandits" that disappear or appear.

A particularly interesting setting is discussed by [Komiyama et al., 2013], where there are lock-up periods when one is required to play the same arm several times in a row. There is an equivalence of that problem to the one studied here. In our setting, we fix the price of the product for an entire day. This is equivalent to a setting where timesteps are taken for each customer, but where there is a lock period over the course of the entire day. In other words, in our scenario, the micro-lock-up periods occur at each step of the game, and their effective lengths are given by $G(t)$. In the work of [Komiyama et al., 2013], ADDEDlock periods are presented but there is no regulating greed based on the size of the lock periods.

The ideas in this paper were inspired by a high scoring entry in the Exploration and Exploitation 3 Phase 1 data mining competition, where the goal was to build a better recommendation system for Yahoo! Front Page news articles. At each time, several articles were available to choose from, and these articles would appear only for short time periods and would never be available again. One of the main ideas in this entry was simple yet effective: if any article gets more than 9 clicks out of the last 100 times we show the article, and keep displaying it until the clickthrough rate goes down. This alone increased the clickthrough rate by almost a quarter of a percent. In the Yahoo! advertising problem, the high reward period was created by the availability of an article (an arm), which is different than the retail store case, but the same effect is present, where regulating the rate of exploitation (i.e., *greed*) over time is beneficial to overall rewards. Here also, it is useful to stop exploring during times when a function like $G(t)$ is high. For Yahoo! Front Page, articles have a short lifespan and some articles are much better than others, in which case, if we find a particularly good article, we should exploit by repeatedly showing that one, and not explore new articles. The framework here distills the problem, allowing us to isolate and study this effect of a time dependent function that we can use to regulate greed over time.

3 Algorithms for regulating greed over time

This section illustrates the problem, the proposed algorithms to regulate greed over time, and theoretical results on the bound on the expected regret of each policy.

3.1 Problem setup

Formally, the stochastic multi-armed bandit problem with regulated greed is a game played in n rounds. At each round t the player chooses an action among a finite set of m possible choices called *arms* (for example, they could be ads shown on a website, recommended videos and articles, or prices). When arm j is played ($j \in \{1, \cdots, m\}$) an *unscaled* random reward $X_j(t)$ is drawn from an unknown distribution and the player receives the *scaled* reward $X_j(t)G(t)$ where $G(t)$ is the *multiplier function*. The distribution of $X_j(t)$ does not change with time (the index t is just used to indicate in which turn the reward was drawn), while $G(t)$ is a known function of time assumed to be bounded (this is, for instance, the number of searches for a particular item on Google or the number of users on Skype). At each turn, the player suffers also a possible regret from not having played the best arm: the mean regret for having played arm j is given by $\Delta_j = \mu_* - \mu_j$, where μ_* is the mean reward of the best arm (indicated by "*") and μ_j is the mean reward obtained when playing arm j. At the end of each turn the player can update her estimate of the mean reward of arm j:

$$\widehat{X}_j = \frac{1}{T_j(t-1)} \sum_{s=1}^{T_j(t-1)} X_j(s), \tag{1}$$

where $T_j(t-1)$ is the number of times arm j has been played before round t starts. This update will hopefully help the player in choosing a good arm in the next round. The total regret at the end of the game is given by

$$R_n = \sum_{t=1}^{n} \sum_{j=1}^{m} \Delta_j G(t) \mathbb{1}_{\{I_t = j\}}, \tag{2}$$

where $\mathbb{1}_{\{I_t=j\}}$ is an indicator function equal to 1 if arm j is played at time t (otherwise its value is 0). The strategies presented in the following sections aim to minimize the expected cumulative regret $\mathbb{E}[R_n]$ by regulating *exploitation* (i.e., *greed*) of the best arm found so far, and *exploration* based on the values of the multiplier function $G(t)$. In general, when the multiplier function is high, the player risks to incur in high regret if a bad arm is played. We show that it is beneficial to stop exploration in this situation and resume exploration when rewards and regrets are lower. A complete list of the symbols used throughout the paper can be found in Appendix E.

3.2 Regulating greed with variable arm pool

In Algorithm 1 we present an algorithm that regulates greed by varying the size m_t of the pool of arms that we are allowed to choose. When the greed function is high, the pool size m_t shrinks, so that we choose randomly among the arms that performed best. When the greed function is low we choose randomly among a larger pool. The size of the pool is given by

$$m_t = \min\left(m, \max\left(1, \frac{cm}{tG(t)}\right)\right). \tag{3}$$

Algorithm 1: variable pool algorithm

```
Input      :number of rounds $n$, number of arms $m$, a constant $c > 1$, and $\{G(t)\}_{t=1}^{n}$;
for $t = m + 1$ to $n$ do
    Set pool size to $m_t = \min\left(m, \max\left(1, \frac{cm}{tG(t)}\right)\right)$;
    Play arm $j$ at random from the pool ;
    Get reward $G(t)X_j$. Update $\widehat{X}_j$ ;
end
```

Let us define

$$\lambda_t = \frac{1}{2m} \sum_{s=1}^{t} \mathbb{1}\left\{\min\left(m, \max\left(1, \frac{cm}{sG(s)}\right)\right) = m\right\},$$

which is half of the number of times that the pool contains all the arms at time t. For the following theorem we require that $\lambda_t > \gamma \log(t)$ for some $\gamma > 5$. If $G(t)$ does not satisfy this requirement, it is easy to construct a new $G'(t)$ from $G(t)$ by setting $G'(t) = (c-1)/t$ for $t \in t, t+1, \cdots, t+2m$ if $\gamma \log(t) > \lceil \gamma \log(t-1) \rceil$, otherwise keep $G'(t) = G(t)$. The following theorem provides a bound on the regret after n rounds.

Theorem 3.1 (variable pool algorithm)**.** *The bound on the mean regret $\mathbb{E}[R_n]$ at time n is given by*

$$\mathbb{E}[R_n] \leq \sum_{t=1}^{n} \sum_{j=1}^{m} \Delta_j G(t) \frac{2}{m_t} \beta_j, \tag{4}$$

where

$$\beta_j(t) = \gamma \log(t)(t)^{-\gamma/5} + \frac{2}{(\Delta_j^{*-m_t})^2} t^{-\frac{\gamma(\Delta_j^{*-m_t})^2}{2}} \tag{5}$$

3.3 Regulating greed with threshold in the ε-greedy algorithm

In Algorithm 2 we present a variation of the ε-greedy algorithm of Auer et al. [2002], in which a threshold z has been introduced in order to regulate greed. An optimal threshold z can also be estimated by running the algorithm on past data and by evaluating the one that gives the lowest regret. At each turn t, when the rewards are "high" (i.e., the $G(t)$ multiplier is above the threshold z) the algorithm exploits the best arm found so far, that is, arm j with the highest mean estimate given in equation (1). When the rewards are "low" (i.e., the $G(t)$ multiplier is under the threshold z), the algorithm will explore with probability $\varepsilon_t = \min\left\{1, \frac{km}{\tilde{t}}\right\}$ an arm at random (each arm has probability $1/m$ of being selected). The number $\tilde{t}$ counts how many times the multiplier function has been under the threshold up to time t, while the constant k is greater than 10 and such that $k > \frac{4}{\min_j \Delta_j}$. The reason of this choice is clear by looking at the expression of $\beta_j(\tilde{t})$ which is a bound on the probability of considering incorrectly a suboptimal arm j being the best choice. By setting the parameter k accordingly, we can ensure the logarithmic bound on the expected cumulative regret over the number of rounds (because the ε_t are $\theta\left(1/t\right)$ and their sum over time is logarithmically bounded, while the $\beta_j(\tilde{t})$ term is $o\left(1/\tilde{t}\right)$).

Algorithm 2: ε-greedy algorithm with regulating threshold

```
Input          :number of rounds $n$, number of arms $m$, threshold $z$, a constant $k > 10$, such that $k > \frac{4}{\min_j \Delta_j}$,
                sequences $\{\varepsilon_t\}_{t=1}^{n} = \min\left\{1, \frac{km}{\tilde{t}}\right\}$ and $\{G(t)\}_{t=1}^{n}$
Initialization :play all arms once and initialize $\widehat{X}_j$ (defined in (1)) for each $j = 1, \cdots, m$
for $t = m + 1$ to $n$ do
    if ($G(t) < z$) then
        with probability $\varepsilon_t$ play an arm uniformly at random (each arm has probability $\frac{1}{m}$ of being selected),
        otherwise (with probability $1 - \varepsilon_t$) play arm $j$ such that
                $\widehat{X}_j \geq \widehat{X}_i \; \forall i$
    else
        play arm $j$ such that
                $\widehat{X}_j \geq \widehat{X}_i \; \forall i$
    end
    Get reward $G(t)X_j$;
    Update $\widehat{X}_j$ ;
end
```

The following theorem provides a bound on the mean regret of this policy (the proof is given in Appendix A).

Theorem 3.2 (ε-greedy algorithm with hard threshold). *The bound on the mean regret $\mathbb{E}[R_n]$ at time n is given by*

$$\begin{aligned}
\mathbb{E}[R_n] \quad \leq \quad & \sum_{j=1}^{m} G(j)\Delta_j & (6)\\
+ \quad & \sum_{t=m+1}^{n} G(t)\mathbb{1}_{\{G(t)<z\}} \sum_{j:\mu_j<\mu_*} \Delta_j \left(\varepsilon_t \frac{1}{m} + (1-\varepsilon_t)\beta_j(\tilde{t})\right) & (7)\\
+ \quad & \sum_{t=m+1}^{n} G(t)\mathbb{1}_{\{G(t)\geq z\}} \sum_{j:\mu_j<\mu_*} \Delta_j\beta_j(\tilde{t}), & (8)
\end{aligned}$$

where

$$\beta_j(\tilde{t}) = k\left(\frac{\tilde{t}}{mke}\right)^{-\frac{k}{10}} \log\left(\frac{\tilde{t}}{mke}\right) + \frac{4}{\Delta_j^2}\left(\frac{\tilde{t}}{mke}\right)^{-\frac{k\Delta_j^2}{4}}. \tag{9}$$

Theorem 3.3 (ε-greedy algorithm with hard threshold). *The bound on the mean regret $\mathbb{E}[R_n]$ at time n is given by*

$$\begin{aligned}
\mathbb{E}[R_n] \quad \leq \quad & \mathcal{O}(1) & (10)\\
+ \quad & \sum_{t=m+1}^{n} G(t)\mathbb{1}_{\{G(t)<z\}} \left(\theta\left(\frac{1}{t}\right) + o\left(\frac{1}{t}\right)\right) & (11)\\
+ \quad & \sum_{t=m+1}^{n} G(t)\mathbb{1}_{\{G(t)\geq z\}} o\left(\frac{1}{t}\right) & (12)
\end{aligned}$$

Intuitively, this bound is better than the usual ε-greedy bound because when $G(t)$ is low it is multiplied by a quantity that is of the order $\theta\left(\frac{1}{t}\right) + o\left(\frac{1}{t}\right)$, while when $G(t)$ is high it is multiplied by a $o\left(\frac{1}{t}\right)$ quantity.

The sum in (6) is the exact mean regret during the initialization phase of Algorithm 2. In (7) we have a bound on the expected regret for turns that present low values of $G(t)$, where the quantity in the parenthesis is the bound on the probability of playing arm j: $\beta_j(\tilde{t})$ is the bound on the probability that arm j is considered being the best arm at round t, and $1/m$ is the probability of choosing arm j when the choice is made at random. Finally, in (8) we have a bound on the expected regret for turns that present high values of $G(t)$ and in this case we consider only the probability $\beta_j(\tilde{t})$ that arm j is the best arm since we do not explore at random during high reward periods. The usual ε-greedy algorithm is a special case when $G(t) = 1\ \forall t$ and $z > 1$. Notice that ε_t is a quantity $\theta\left(1/\tilde{t}\right)$, while $\beta_j(\tilde{t})$ is $o\left(1/\tilde{t}\right)$, so that an asymptotic logarithmic bound in n holds for $\mathbb{E}[R_n]$ if $\tilde{t}$ grows at the same rate as t (because of the logarithmic bound on the harmonic series).

We want to compare this bound with the one of the usual version of the ε-greedy algorithm but, since the old version is not well suited for the setting in which the rewards are altered by the multiplier function, we discount the rewards obtained at each round (by simply dividing them by $G(t)$) so that it can also produce accurate estimates of the mean reward for each arm. This "smarter" version of the ε-greedy algorithm is presented in Algorithm 7 (Section 4). The bound on the probability of playing a suboptimal arm j for the usual ε-greedy algorithm is given by $\beta_j(t)$ (i.e. $\beta_j(\tilde{t})$ when $\tilde{t} = t$) and we refer to it as $\beta_j^{\text{old}}(t)$. In general, $\beta_j^{\text{old}}(t)$ is lower than $\beta_j(\tilde{t})$ (since $\tilde{t} \leq t$). Intuitively, this reflects the fact that the new algorithm performs fewer exploration steps. Moreover, in the usual ε-greedy algorithm, the probability of choosing arm j at time t is given by

$$\mathbb{P}\left(\{I_t^{\text{old}} = j\}\right) = \varepsilon_t \frac{1}{m} + (1-\varepsilon_t)\beta_j^{\text{old}}(t),$$

which is less than the probability of the new algorithm in case of low $G(t)$

$$\mathbb{P}\left(\{I_t^{\text{new}} = j\}\right) = \varepsilon_t \frac{1}{m} + (1-\varepsilon_t)\beta_j(\tilde{t}),$$

but can easily be higher than the probability of the new algorithm in case of high rewards (which is given by only $\beta_j(\tilde{t})$). In fact,

$$\begin{aligned}\mathbb{P}\left(\{I_t^{\text{old}} = j\}\right) - \mathbb{P}\left(\{I_t^{\text{new}} = j\}\right) &= \varepsilon_t \frac{1}{m} + (1-\varepsilon_t)\beta_j^{\text{old}}(t) - \beta_j(\tilde{t}) \\ &= \frac{1}{m}\min\left\{1, \frac{km}{t}\right\} - \beta_j^{\text{old}}(t)\min\left\{1, \frac{km}{t}\right\} + \beta_j^{\text{old}}(t) - \beta_j(\tilde{t}),\end{aligned}$$

if $t > km$ we get

$$\frac{1}{m}\frac{km}{t} + \beta_j^{\text{old}}(t)\left(1 - \frac{km}{t}\right) - \beta_j(\tilde{t}), \tag{13}$$

if $t \leq km$ we get

$$\frac{1}{m} - \cancel{\beta_j^{\text{old}}(t)} + \cancel{\beta_j^{\text{old}}(t)} - \beta_j(\tilde{t}), \tag{14}$$

and for t large enough both expressions are positive since $\beta_j(\tilde{t})$ is $o\left(1/\tilde{t}\right)$ and we assume that $\tilde{t}$ is $\theta(t)$. Having (13) and (14) positive means that if we are in a high-rewards period the probability of choosing a suboptimal arm decreases faster in Algorithm 2. In that case, Algorithm 2 would have lower regret than the ε-greedy algorithm.
In practice, the threshold z should be defined as $\text{argmin}(\mathbb{E}[R_n])$. If this is too computationally challenging, but past data are available, a good value for z can be chosen using cross validation techniques, i.e. by trying different thresholds with the available data and by choosing the one that yields the best performance.

The following Corollary illustrate the benefits of the bound in a simple scenario when the multiplier function can only take two values and the regulating threshold divides the higher value from the lower one.

Corollary 3.1. *Suppose the greed function $G(t)$ takes only two values: g_{low} and g_{high}. At each turn t it takes the value g_{low} for a fraction q of the turns played, and the value g_{high} for the remaining $t - qt$ turns (for example, if $q = 1/2$, $G(t)$ alternates at each turn between g_{low} and g_{high}). Then, the bound on the expected regret at turn n reduces to:*

$$\begin{aligned}\mathbb{E}[R_n] &\leq \mathcal{O}(1) \\ &+ \frac{k}{q}\Delta_{tot}\, g_{low} \sum_{t=m+1}^{n} \mathbb{1}_{\{G(t)=g_{low}\}}\left(\frac{1}{t} + o\left(\frac{1}{t}\right)\right) \\ &+ \Delta_{tot}\, g_{high} \sum_{t=m+1}^{n} \mathbb{1}_{\{G(t)=g_{high}\}}\, o\left(\frac{1}{t}\right),\end{aligned}$$

where $\Delta_{tot} = \sum_{j:\mu_j<\mu_*} \Delta_j$.

The term that hurts regret the most $(1/t)$ is multiplied only by g_{low}, and not by g_{high}. When the rewards are high (and so is the possible regret), only terms of order $o(1/t)$ are present. If exploration were permitted during the high reward zone, there would have been large terms of g_{high}/t, which is what the algorithm is designed to avoid.

3.4 Soft ε-greedy algorithm

We present in Algorithm 3 a "soft version" of the ε-greedy algorithm where greed is regulated gradually (in contrast with the hard threshold of the previous section). Again, in high reward zones, exploitation will be preferred, while in low reward zones the algorithm will explore the arms more. Let us define the following function

$$\psi(t) = \frac{\log\left(1 + \frac{1}{G(t)}\right)}{\log\left(1 + \frac{1}{\min_{s\in\{m+1,\cdots,n\}} G(s)}\right)}, \tag{15}$$

and let $\gamma = \min_{s\in\{m+1,\cdots,n\}} \psi(s)$. Notice that $0 < \psi(t) \leq 1\ \forall t$ and that its values are close to 0 when $G(t)$ is high, while they are close to 1 for low values of $G(t)$. The new probabilities of exploration during the game are given at each turn t by $\varepsilon_t = \min\left\{\psi(t), \frac{km}{t}\right\}$. In this way, we still maintain the linear decay of the probabilities of exploration, but we push them to zero to avoid high regrets when the multiplier function $G(t)$ is high. We generally assume that $\min_{s\in\{m+1,\cdots,n\}} G(s)$ is not smaller than 1. The usual case is recovered when $G(t) = 1$ for all t.

The following theorem (proved in Appendix B) shows that a logarithmic bound holds in this case too (because the ε_t are $\theta\left(1/t\right)$ and their sum over time is logarithmically bounded, while the $\beta_j^S(t)$ term is $o\left(1/t\right)$).

```
Algorithm 3: Soft $\varepsilon$-greedy algorithm
Input          :number of rounds $n$, number of arms $m$, a constant $k > 10$, such that $k > \frac{4}{\min_j \Delta_j}$, sequences
                $\{\varepsilon_t\}_{t=1}^n = \min\left\{\psi(t), \frac{km}{t}\right\}$ and $\{G(t)\}_{t=1}^n$
Initialization :play all arms once and initialize $\widehat{X}_j$ (defined in (1)) for each $j = 1, \cdots, m$
for $t = m+1$ to $n$ do
    With probability $\varepsilon_t$ play an arm uniformly at random (each arm has probability $\frac{1}{m}$ of being selected),
    otherwise (with probability $1 - \varepsilon_t$) play arm $j$ such that
                                  $\widehat{X}_j \geq \widehat{X}_i \ \forall i$
    Get reward $G(t)X_j$;
    Update $\widehat{X}_j$;
end
```

Theorem 3.4 (Regret-bound for soft-ε-greedy algorithm). *The bound on the mean regret $\mathbb{E}[R_n]$ at time n is given by*

$$\mathbb{E}[R_n] \leq \sum_{j=1}^{m} G(j)\Delta_j \tag{16}$$

$$+ \sum_{t=m+1}^{n} G(t) \sum_{j:\mu_j<\mu_*} \Delta_j \left(\varepsilon_t \frac{1}{m} + (1-\varepsilon_t)\beta_j^S(t)\right) \tag{17}$$

where

$$\beta_j^S(t) = k\left(\frac{\gamma t}{mke}\right)^{-\frac{k}{10}} \log\left(\frac{\gamma t}{mke}\right) + \frac{4}{\Delta_j^2}\left(\frac{\gamma t}{mke}\right)^{-\frac{k\Delta_j^2}{4}}. \tag{18}$$

Theorem 3.5 (Regret-bound for soft-ε-greedy algorithm). *The bound on the mean regret $\mathbb{E}[R_n]$ at time n is given by*

$$\mathbb{E}[R_n] \leq \mathcal{O}(1) + \sum_{t=m+1}^{n} G(t)\left(\theta\left(\frac{1}{t}\right) + o\left(\frac{1}{t}\right)\right) \tag{19}$$

Intuitively, when $G(t)$ is low it is multiplied by a $\theta\left(\frac{1}{t}\right)$ quantity, while when $G(t)$ is high it is multiplied by a $o\left(\frac{1}{t}\right)$ quantity.

The sum in (16) is the exact mean regret during the initialization of Algorithm 3. For the rounds after the initialization phase, the quantity in the parenthesis of (17) is the bound on the probability of playing arm j (where $\beta_j^S(t)$ is the bound on the probability that arm j is the best arm at round t, and $1/m$ is the probability of choosing arm j when the choice is made at random).

As before, we want to compare this bound with the "smarter" version of the ε-greedy algorithm presented in Algorithm 7. In the usual ε-greedy algorithm, after the "critical time" $n' = km$, the probability $\mathbb{P}(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{i,T_i(t-1)})$ of arm j being the current best arm, can be bounded by a quantity $\beta_j^{\text{old}}(t)$ that is $o(1/t)$ as t grows. Before time n', the decay of $\mathbb{P}(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{i,T_i(t-1)})$ is faster and the bound is a quantity that is $o\left(1/t^{\lambda}\right), \forall\lambda$ as t grows (see Remark 1 in Appendix A). The probability of choosing a suboptimal arm j changes as follows:

- if $t < n'$, $\mathbb{P}(\{I_t = j\}) = \frac{1}{m}$;
- if $t \geq n'$, $\mathbb{P}(\{I_t = j\}) = \frac{k}{t} + \left(1 - \frac{km}{t}\right)\beta_j^{\text{old}}(t)$, which is $\theta\left(\frac{1}{t}\right)$ as t grows.

In the soft-ε-greedy algorithm, before time w defined as $w = \operatorname{argmin} f(s)$, subject to $f(s) < \gamma$, where $f(s) = \frac{km}{s}$, we have that $\beta_j^S(t)$, which is the bound on the probability $\mathbb{P}(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{i,T_i(t-1)})$ of arm j being the current best arm, is a quantity that is $o\left(1/(\gamma t)^{\lambda}\right), \forall\lambda$ as t grows (the argument is similar to the Remark 1 in Appendix A). After w, it can be bounded by a quantity that is $o(1/(\gamma t))$ as t grows. The probability of choosing a suboptimal arm j changes as follows:

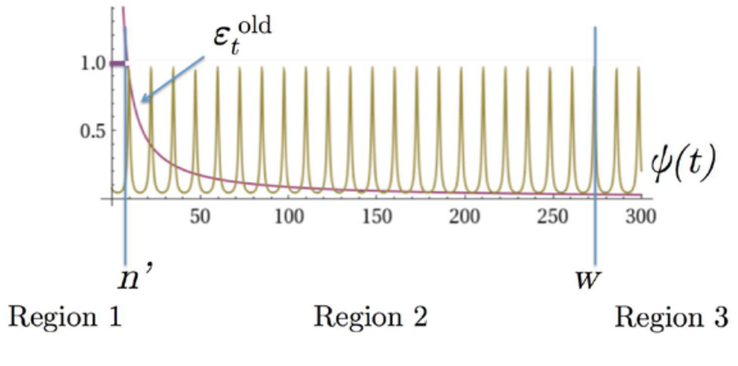
- if $t < n'$, $\mathbb{P}(\{I_t = j\}) = \frac{1}{m}\psi(t) + (1 - \psi(t))\beta_j^S(t)$;
- if $n' \le t \le w$, $\mathbb{P}(\{I_t = j\}) = \frac{1}{m}\min\left\{\psi(t), \frac{km}{t}\right\} + \left(1 - \min\left\{\psi(t), \frac{km}{t}\right\}\right)\beta_j^S(t)$;
- if $t > w$, $\mathbb{P}(\{I_t = j\}) = \frac{k}{t} + \left(1 - \frac{k}{t}\right)\beta_j^S(t)$.

In order to interpret these quantities, let us see what happens for high or low values of the multiplier $G(t)$ as t grows in Table 1. For brevity, we abuse notation when using Landau's symbols, because in some cases t is not allowed to go to infinity; it is convenient to still use the "little o" notation to compare the decay rates of the probabilities of choosing a suboptimal arm, which also gives a qualitative explanation of what happens when using the algorithms. For the soft-ε-algorithm, the rate at which the probability of choosing a suboptimal arm decays is faster when $G(t)$ is high, and worse when $G(t)$ is low. Notice that the parameter γ slows down the decay with respect to the usual ε-greedy algorithm. This is direct consequence of the slower exploration. An example of a typical behavior of $\psi(t)$ and $\varepsilon_t^{\text{old}}$ is shown in Figure 2, where $G(t) = 20 + 19\sin(t/2)$.

Table 1: Summary of the decay rate of the probabilities of choosing a suboptimal arm for the soft-ε-greedy algorithm and the usual ε-greedy algorithm (supposing it is taking in account the time-patterns.) The decay depends on the time-regions of the game presented in Figure 2.

Region	round	$G(t)$	$\mathbb{P}(\{I_t = j\})^{\text{old}}$	$\mathbb{P}(\{I_t = j\})^{\text{soft}}$	$\mathbb{P}(\{I_t = j\})^{\text{soft}} < \mathbb{P}(\{I_t = j\})^{\text{old}}$?
1	$t < n'$	high	$\frac{1}{m}$	$o\left(\frac{1}{(\gamma t)^\lambda}\right), \forall\lambda$	yes, much better
		low	$\frac{1}{m}$	close to $\frac{1}{m}$	no, but not by much
2	$n' \le t \le w$	high	$\theta\left(\frac{1}{t}\right) + o\left(\frac{1}{t}\right)$	$o\left(\frac{1}{(\gamma t)^\lambda}\right), \forall\lambda$	yes, much better
		low	$\theta\left(\frac{1}{t}\right) + o\left(\frac{1}{t}\right)$	$\theta\left(\frac{1}{t}\right) + o\left(\frac{1}{(\gamma t)^\lambda}\right), \forall\lambda$	yes, but not by much
3	$t > w$	high	$\theta\left(\frac{1}{t}\right) + o\left(\frac{1}{t}\right)$	$\theta\left(\frac{1}{t}\right) + o\left(\frac{1}{\gamma t}\right)$	no, but not by much
		low	$\theta\left(\frac{1}{t}\right) + o\left(\frac{1}{t}\right)$	$\theta\left(\frac{1}{t}\right) + o\left(\frac{1}{\gamma t}\right)$	no, but not by much

Figure 2: Comparison of probabilities of exploration over the number of rounds. Before n', $\varepsilon_t^{\text{old}}$ is 1 and always greater than $\psi(t)$. After w, $\varepsilon_t^{\text{old}}$ is always less than $\psi(t)$.



3.5 Regulating greed in the UCB algorithm

Following what has been presented to improve the ε-greedy algorithm in this setting, we introduce in Algorithm 4 a modification of the UCB algorithm. We again set a threshold z and, if the multiplier of the rewards $G(t)$ is above this level, the new algorithm exploits the best arm. When $G(t)$ is under the threshold, the algorithm is going to play the arm with the highest upper confidence bound on the mean estimate.

Algorithm 4: UCB algorithm with regulating threshold

```
Input          : number of rounds $n$, number of arms $m$, threshold $z$, sequence $\{G(t)\}_{t=1}^{n}$
Initialization : play all arms once and initialize $\widehat{X}_j$ (as defined in (1)) for each $j = 1, \cdots, m$
for $t = m + 1$ to $n$ do
    if $(G(t) < z)$ then
        play arm $j$ with the highest upper confidence bound on the mean estimate
            $$\widehat{X}_{j,T_j(t-1)} + \sqrt{\frac{2\log t}{T_j(t-1)}};$$
    else
        play arm $j$ such that
            $$\widehat{X}_j \geq \widehat{X}_i \ \forall i;$$
    end
    end
    Get reward $G(t)X_j$;
    Update $\widehat{X}_j$;
end
```

It is possible to prove that also in this case the regret can be bounded logarithmically in n. Let $B = \{t : G(t-1) < z, G(t) > z\}$ be the set of rounds where the high-reward zone is entered, and let τ_t be the last round of the high-reward zone that was entered at time t. Let us call $y_1, y_2, \cdots, y_B$ the elements of B and order them in increasing order such that $y_1 < y_2 < \cdots < y_B$. Let us also define for every $k \in \{1, \cdots, |B|\}$ the set $Y_k = \{t : t \geq y_k, G(t) > z, t < y_{k+1}\}$ (where $y_{B+1} = n$) of times in the high-reward period entered at time y_k, and let $\Lambda_k = \max_{t \in Y_k} G(t)$ the highest value of $G(t)$ on Y_k. Finally, for every k, let $R_k = \Lambda_k |Y_k|$.

Now, given a game of n total rounds, we can "collapse" the kth high reward zone into the entering time y_k by defining $G(y_k) = R_k$, for all k. Now, the maximum regret over B is given by $(\max_j \Delta_j) \sum_{k=1}^{|B|} R_k$. By eliminating the set B from the game, we have transformed the original game into a shorter one, with η steps, where $G(t)$ is bounded by z and the usual UCB algorithm is played. When the size of set B decreases with n, (is of order $\theta(1/t)$ after an arbitrary time), the total regret has a logarithmic bound in n.

The ε-greedy methods are more amenable to this type of analysis than UCB methods, because the proofs require bounds on the probability of choosing the wrong arm *at each turn*. The UCB proof instead require us to bound the expected number of times the suboptimal arms are played, without regard to *when* those arms were chosen. We were able to avoid using the maximum of the $G(t)$ values in the ε-greedy proofs, but this is unavoidable in the UCB proofs without leaving terms in the bound that cannot be explicitly calculated or simplified (an alternate proof would use weaker Central Limit Theorem arguments).

Theorem 3.6 (Regret-bound for the regulated UCB algorithm)**.** *The bound on the mean regret $\mathbb{E}[R_n]$ at time n is given by*

$$\begin{aligned} \mathbb{E}[R_n] &\leq \sum_{j=1}^{m} G(j)\Delta_j && (20)\\ &+ z\left[8\sum_{j:\mu_j<\mu_*}\left(\frac{\log\eta}{\Delta_j}\right)+\left(1+\frac{\pi^2}{3}\right)\left(\sum_{j=1}^{m}\Delta_j\right)\right] && (21)\\ &+ (\max_j \Delta_j)\sum_{k=1}^{|B|} R_k && (22)\end{aligned}$$

Theorem 3.7 (Regret-bound for the regulated UCB algorithm)**.** *The bound on the mean regret $\mathbb{E}[R_n]$ at time n is given by*

$$\begin{aligned} \mathbb{E}[R_n] &\leq \mathcal{O}(1) && (23)\\ &+ z\,\mathcal{O}(\log(\eta))+\mathcal{O}(1) && (24)\\ &+ \mathcal{O}(1) && (25)\end{aligned}$$

The first sum in (20) is the exact mean regret of the initialization phase of Algorithm 4, the third sum in (22) is the bound on the regret from the high-reward zones that have been collapsed, and the second term in (21) is the bound on the regret for η rounds when $G(t)$ is under the threshold z and it follows from the usual bound on the UCB algorithm (for n rounds the UCB algorithm has a mean regret bounded by $\sum_{j=1}^{m}\frac{\log n}{\Delta_j}+\left(1+\frac{\pi^2}{3}\right)\sum_{j=1}^{m}\Delta_j$).
Again, the threshold z should be defined as $\text{argmin}(\mathbb{E}[R_n])$ or, if past data are available, z can be chosen using cross validation.

3.6 The soft UCB algorithm

In Algorithm 5, present now a "soft version" of the UCB algorithm where greed is regulated gradually (in contrast with the hard threshold of the previous section). Again, in high reward zones, exploitation will be preferred, while in low reward zones the algorithm will explore the arms.
Let us define the following function:

$$\xi(t)=\left(1+\frac{t}{G(t)}\right). \qquad (26)$$

At each turn t of the game, the algorithm plays the arm with the highest upper confidence bound on the mean estimate, but, with the introduction of $\xi(t)$, the confidence interval around $\widehat{X}_{j,T_j(t-1)}$ is built in a way such that, when $G(t)$ is high, it collapses on the estimate itself, forcing the player to choose the arm with the highest mean estimate (thus, leading to a pure exploitation policy). In contrast, when the multiplier $G(t)$ is low, the confidence interval around $\widehat{X}_{j,T_j(t-1)}$ stretches out, making the player explore more easily arms with high uncertainty.
One of the main difficulties of the formulation of these bounds is to define a correct functional form for $\xi(t)$ so that it is possible to obtain smoothness in the arm decision, reasonable Chernoff-Hoeffding inequality bounds while working out the proof (see Appendix C), and a convergent series (the second summation in (28)).
Also in this case, it is possible to achieve a bound that grows logarithmically in n.

Theorem 3.8 (Regret-bound for soft-UCB algorithm)**.** *The bound on the mean regret $\mathbb{E}[R_n]$ at time n is given by*

$$\begin{aligned} \mathbb{E}[R_n] &\leq \sum_{j=1}^{m} G(j)\Delta_j && (27)\\ &+ \max_{t\in\{m+1,\cdots,n\}} G(t)\left[\sum_{j:\mu_j<\mu_*}\frac{8}{\Delta_j}\log\left(\max_{t\in\{m+1,\cdots,n\}}\xi(t)\right)+\sum_{j=1}^{m}\Delta_j\left(1+\sum_{t=m+1}^{n}2\xi(t)^{-4}(t-1-m)^2\right)\right]. && (28)\end{aligned}$$

Algorithm 5: Soft UCB algorithm

```
Input          :number of rounds $n$, number of arms $m$, sequence $\{G(t)\}_{t=1}^{n}$
Initialization :play all arms once and initialize $\widehat{X}_j$ (as defined in (1)) for each $j = 1, \cdots, m$
for $t = m + 1$ to $n$ do
    play arm $j$ with the highest upper confidence bound on the mean estimate:
        $\widehat{X}_{j,T_j(t-1)} + \sqrt{\frac{2 \log \xi(t)}{T_j(t-1)}}$;
    Get reward $G(t)X_j$;
    Update $\widehat{X}_j$;
end
```

Theorem 3.9 (Regret-bound for soft-UCB algorithm). *The bound on the mean regret* $\mathbb{E}[R_n]$ *at time* n *is given by*

$$\mathbb{E}[R_n] \leq \mathcal{O}(1) \tag{29}$$

$$+ \max_{t\in\{m+1,\cdots,n\}} G(t) \left[\mathcal{O}\left(\log\left(\max_{t\in\{m+1,\cdots,n\}} \xi(t)\right)\right) + \mathcal{O}(1)\right] \tag{30}$$

The first sum in (27) is the exact mean regret of the initialization phase of Algorithm 5. For the rounds after the initialization phase, the mean regret is bounded by the quantity in (28), which is almost identical to the bound of the usual UCB algorithm if we assume $G(t) = 1$ (i.e., rewards are not modified by the multiplier function).

4 Experimental results

We consider three types of multiplier function $G(t)$:

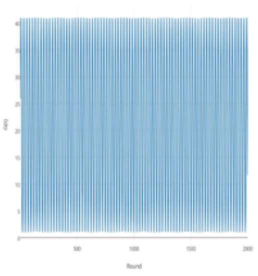
- The *Wave Greed* (Figure 3a): in this case customers come in waves: $G(t) = 21 + 20\sin(0.25t)$. We want to exploit the best arm found so far during the peaks, and explore the other arms during low-rewards periods ;
- The *Christmas Greed* (Figure 3b): again, $G(t) = 21 + 20\sin(0.25t)$, but when $t \in \{650, 651, \cdots, 670\}$, $G(t) = 1000$ which shows that there is a peak in the rewards offered by the game (which we call "Christmas", in analogy to the phenomenon of the boom of customers during the Christmas holidays) ;
- The *Step Greed* (Figure 3c): this case is similar to the *Wave Greed* case, but this time the function is not smooth: $G(t) = 200$, but for $t \in \{600, 601, \cdots, 800\} \cup \{1000, 1001, \cdots, 1200\} \cup \{1400, 1402, \cdots, 1600\}$ we have $G(t) = 400$.

We consider a game with 500 arms and normally distributed rewards. Each arm $j \in \{1, \cdots, 500\}$ has mean reward $\mu_j = 0.1 + (200 + 1.5(500 - j + 1))/(1.5 \times 500)$ and common standard deviation $\sigma = 0.05$. The arms were chosen in this way so that X_j would take values (with high probability) in $[0, 1]$. Having a bounded support for X_j is a standard assumption made when proving regret bounds (see Auer et al. [2002]). We play 2000 rounds each game. After 2000 rounds the algorithms all essentially have determined which arm is the best and tend to perform very similarly from that point onwards.

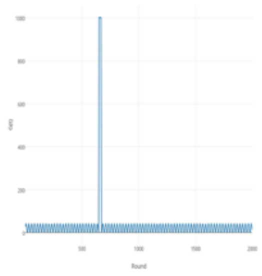
The well-known UCB and ε-greedy algorithms are not suitable for the setting in which the rewards are altered by the multiplier function. Thus, in their current form, we can not compare directly with them. The fact that rewards are multiplied would irremediably bias all the estimations of the mean rewards, leading UCB and ε-greedy to choose arms that look good just because they happened to be played in a high reward period. For example, suppose we show an ad on a website at lunch time: many people will see it because at that time the web-surfing is at its peak (i.e., the $G(t)$ multiplier is high). So even if the ad was bad, we may register more clicks than a good ad showed at 3:00AM (i.e., the $G(t)$ multiplier is low). To obtain a fair comparison, we created "smarter" versions of the UCB and ε-greedy algorithms in which the rewards are discounted at each round (by simply dividing them by $G(t)$) so that also the old version of the algorithms can be smarter in that they can produce accurate estimates of the mean reward for each arm. The smarter version of the usual UCB algorithm is presented in Algorithm 6 and the one for the ε-greedy algorithm is shown in Algorithm 7. For the three multiplier functions, we report the performance of the algorithms in Figures 4, 5, and 6.

Figure 3: Shapes of the multiplier functions used in the experiments.

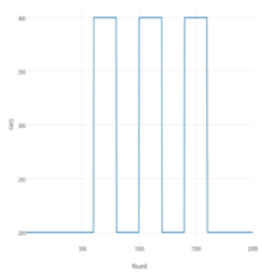
(a) The *Wave Greed*



(b) The *Christmas Greed*



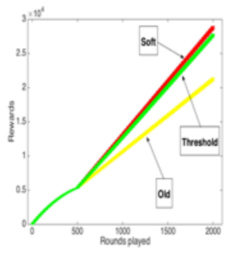
(c) The *Step Greed*



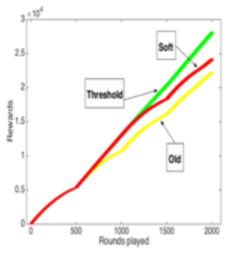
In Figures 7, 8, and 9, we change the rewards to have a Bernoulli distribution (the assumption of bounded support is verified). Similarly to the normal case, each arm $j \in \{1, \cdots, 500\}$ has probability of success $p_j = 0.1 + (200 + 1.5(500 - j + 1))/(1.5 \times 500)$. One of the advantages of the ε-greedy algorithm is that there are no assumptions on the distribution of the rewards, while in UCB they need bounded support ($[0, 1]$ for convenience, so it is easier to use Hoeffding's inequality).

Figure 4: Comparison for the *Wave Greed* case.

(a) ε-greedy algorithms rewards.



(b) UCB algorithms rewards.



(c) Final rewards comparison

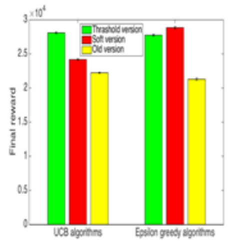
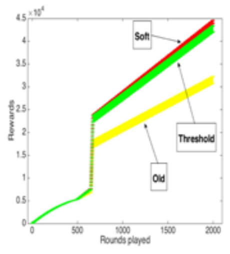
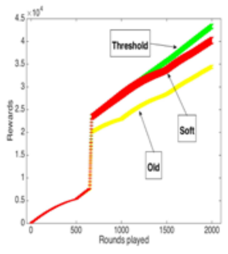


Figure 5: Comparison for the *Christmas Greed* case.

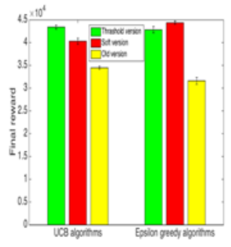
(a) ε-greedy algorithms rewards.



(b) UCB algorithms rewards.



(c) Final rewards comparison



Algorithm 6: Smarter version of the usual UCB algorithm

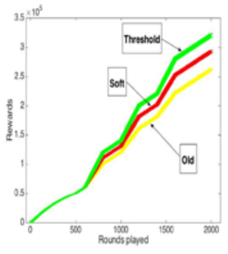
```
Input          : number of rounds $n$, number of arms $m$, sequence $\{G(t)\}_{t=1}^{n}$
Initialization : play all arms once and initialize $\widehat{X}_j$ (as defined in (1)) for each $j = 1, \cdots, m$
for $t = m + 1$ to $n$ do
    play arm $j$ with the highest upper confidence bound on the mean estimate:
        $$\widehat{X}_{j,T_j(t-1)} + \sqrt{\frac{2\log(t)}{T_j(t-1)}}$$
    Get reward $G(t)X_j$;
    Update $\widehat{X}_j$;
end
```

Algorithm 7: Smarter version of the usual ε-greedy algorithm

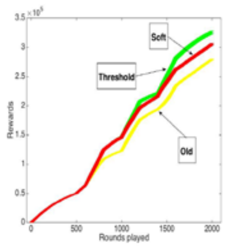
```
Input          : number of rounds $n$, number of arms $m$, a constant $c > 10$, a constant $d$ such that $d < \min_j \Delta_j$ and
                 $0 < d < 1$, sequences $\{\varepsilon_t\}_{t=1}^{n} = \min\left\{1, \frac{cm}{d^2 t}\right\}$ and $\{G(t)\}_{t=1}^{n}$
Initialization : play all arms once and initialize $\widehat{X}_j$ (as defined in (1)) for each $j = 1, \cdots, m$
for $t = m + 1$ to $n$ do
    with probability $\varepsilon_t$ play an arm uniformly at random (each arm has probability $\frac{1}{m}$ of being selected),
    otherwise (with probability $1 - \varepsilon_t$) play arm $j$ such that
        $$\widehat{X}_j \geq \widehat{X}_i \ \forall i$$
    Get reward $G(t)X_j$;
    Update $\widehat{X}_j$;
end
```

Figure 6: Comparison for the *Step Greed* case.

(a) ε-greedy algorithms rewards.



(b) UCB algorithms rewards.



(c) Final rewards comparison.

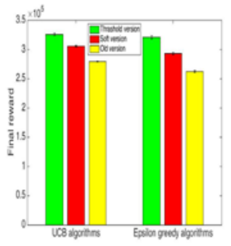
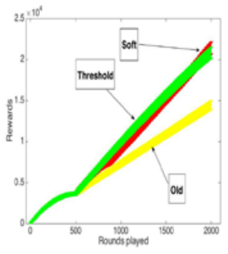
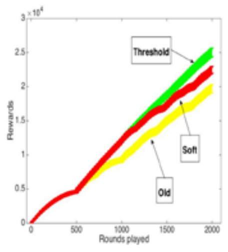


Figure 7: Comparison for the *Wave Greed* case.

(a) ε-greedy algorithms rewards.



(b) UCB algorithms rewards.



(c) Final rewards comparison.

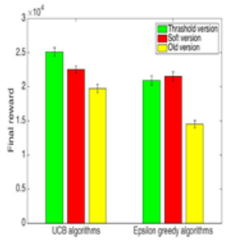
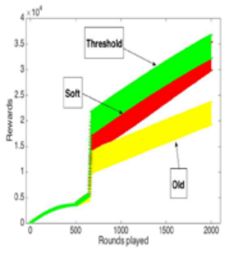
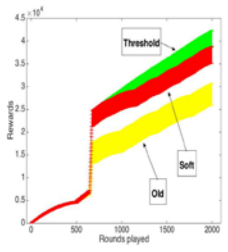


Figure 8: Comparison for the *Christmas Greed* case.

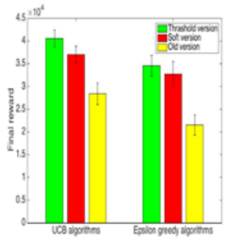
(a) ε-greedy algorithms rewards.



(b) UCB algorithms rewards.



(c) Final rewards comparison.

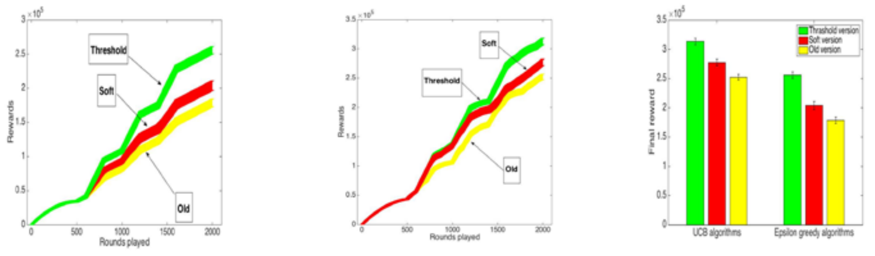


4.1 Discussion on Yahoo! contest

The motivation of this work comes from a high scoring entry in the Exploration and Exploitation 3 contest, where the goal was to build a better recommender system for Yahoo! Front Page news article recommendations. The contest data, which was from Yahoo! and allows for unbiased evaluations, is described by Li et al. [2010]. These data had several challenging characteristics, including broad trends over time in click through rate, arms (news articles) appearing and disappearing over time, the inability to access the data in order to cross-validate, and other complexities. This paper does not aim to handle all of these, but only the one which led to a key insight in increased performance, which is the regulation of greed over time. Although there were features available for each time, none of the contestants were able to successfully use the features to substantially boost performance, and the exploration/exploitation aspects turned out to be more important. Here are the main

Figure 9: Comparison for the *Step Greed* case.

(a) ε-greedy algorithms rewards. (b) UCB algorithms rewards. (c) Final rewards comparison.



insights leading to large performance gains, all involving regulating greed over time:

- "Peak grabber": Stop exploration when a good arm appears. Specifically, when the article was clicked 9/100 times, keep showing it and stop exploration all together until the arm's click through rate drops below that of another arm. Since this strategy does not handle the massive global trends we observed in the data, it needed to be modified as follows:
- "Dynamic peak grabber": Stop exploration when the click through rate of one arm is at least 15% above that of the global click through rate.
- Stop exploring old articles: We can determine approximately how long the arm is likely to stay, and we reduce exploration gradually as the arm gets older.
- Do not fully explore new arms: When a new arm appears, do not use 1 as the upper confidence bound for the probability of click, which would force a UCB algorithm to explore it, use .88 instead. This allows the algorithm to continue exploiting the arms that are known to be good rather than exploring new ones.

The peak grabber strategies inspired the abstracted setting here, where one can think of a good article appearing during periods of high $G(t)$, where we would want to limit exploration; however, the other strategies are also relevant cases where the exploration/exploitation tradeoff is regulated over time. There were no "lock-up" periods in the contest dataset, though as discussed earlier, the $G(t)$ function is also relevant for modeling that setting. The large global trends we observed in the contest data click through rates are very relevant to the $G(t)$ model, since obviously one would want to explore less when the click rate is high in order to get more clicks overall.

5 Conclusions

The dynamic trends we observe in most retail and marketing settings are dramatic. It is possible that understanding these dynamics and how to take advantage of them is central to the success of multi-armed bandit algorithms. We showed in this work how to adapt regret bound analysis to this setting, where we now need to consider not only how many times an arm was pulled in the past, but precisely when the arm was pulled. The key element of our algorithms is that they regulate greed (exploitation) over time, where during high reward periods, less exploration is performed.

There are many possible extensions to this work. In particular, if $G(t)$ is not known in advance, it may be easy to estimate from data in real time, as in the dynamic peak grabber strategy. The analysis of the algorithms in this paper could be extended to other important multi-armed bandit algorithms besides ε-greedy and UCB. Further,future work will consider the connection of mortal bandits (with appearing/disappearing arms) with the $G(t)$ setting, since for mortal bandits, each bandit's $G(t)$ function can change at a different rate.

Acknowledgments

We thank team members Ed Su and Virot Ta Chiraphadhanakul for their work on the ICML Exploration and Exploitation 3 contest that inspired the work here. Thank you also to Philippe Rigollet for helpful comments and encouragement. This project was partially funded by the Ford-MIT alliance.

References

Oren Anava, Elad Hazan, Shie Mannor, and Ohad Shamir. Online learning for time series prediction. *arXiv preprint arXiv:1302.6927*, 2013.

Jean-Yves Audibert, Sébastien Bubeck, et al. Best arm identification in multi-armed bandits. *Conference on Learning Theory 2010-Proceedings*, 2010.

Peter Auer and Ronald Ortner. UCB revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, 61(1-2):55–65, 2010.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002. 3.3, 4, A

Donald A Berry and Bert Fristedt. *Bandit Problems: Sequential Allocation of Experiments (Monographs on Statistics and Applied Probability)*. Springer, 1985.

Omar Besbes, Yonatan Gur, and Assaf Zeevi. Optimal exploration-exploitation in a multi-armed-bandit problem with non-stationary rewards. *Available at SSRN 2436629*, 2014. 2

Olivier Bousquet. New approaches to statistical learning theory. *Annals of the Institute of Statistical Mathematics*, 55(2): 371–389, 2003. ISSN 0020-3157. doi: 10.1007/BF02530506. URL `http://dx.doi.org/10.1007/BF02530506`.

Sébastien Bubeck and Nicolo Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *arXiv preprint arXiv:1204.5721*, 2012.

Deepayan Chakrabarti, Ravi Kumar, Filip Radlinski, and Eli Upfal. Mortal multi-armed bandits. In *Advances in Neural Information Processing Systems*, pages 273–280, 2009. 2

Aurélien Garivier and Olivier Cappé. The KL-UCB algorithm for bounded stochastic bandits and beyond. *arXiv preprint arXiv:1102.2490*, 2011.

Aurélien Garivier and Eric Moulines. On upper-confidence bound policies for non-stationary bandit problems. *arXiv preprint arXiv:0805.3415*, 2008. 2

Emilie Kaufmann, Olivier Cappé, and Aurélien Garivier. On bayesian upper confidence bounds for bandit problems. In *International Conference on Artificial Intelligence and Statistics*, pages 592–600, 2012.

Junpei Komiyama, Issei Sato, and Hiroshi Nakagawa. Multi-armed bandit problem with lock-up periods. In *Asian Conference on Machine Learning*, pages 116–132, 2013. 2

Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, pages 661–670. ACM, 2010. 4.1

Haoyang Liu, Keqin Liu, and Qing Zhao. Learning in a changing world: Restless multiarmed bandit with unknown dynamics. *IEEE Transactions on Information Theory*, 59(3):1902–1916, 2013. 2

Herbert Robbins. Some aspects of the sequential design of experiments. In *Herbert Robbins Selected Papers*, pages 169–177. Springer, 1985.

Cynthia Rudin, Virot Ta Chiraphadhanakul, and Edward Su. Regulating greed over time for Yahoo! front page news article recommendations, from the Exploration and Exploitation 3 Challenge. lecture slides, 2012.

Aleksandrs Slivkins and Eli Upfal. Adapting to a stochastically changing environment: The dynamic multi-armed bandits problem. Technical Report CS-07-05, Brown University, 2007. 2

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, pages 285–294, 1933.

A Regret-bound for ε-greedy algorithm with hard threshold

The regret at round n is given by

$$R_n = \sum_{t=1}^{n}\sum_{j=1}^{m} \Delta_j G(t) \mathbb{1}_{\{I_t=j\}}, \tag{31}$$

where $G(t)$ is the greed function evaluated at time t, $\mathbb{1}_{\{I_t=j\}}$ is an indicator function equal to 1 if arm j is played at time t (otherwise its value is 0) and $\Delta_j = \mu^* - \mu_j$ is the difference between the mean of the best arm reward distribution and the mean of the j's arm reward distribution. By considering the threshold z which determines which rule is applied to decide what arm to play, we can rewrite the regret as

$$\begin{aligned} R_n &= \sum_{t=1}^{n}\sum_{j=1}^{m} \Delta_j G(t) \mathbb{1}_{\{G(t)<z\}} \mathbb{1}_{\{I_t=j\}} + \\ &+ \sum_{t=1}^{n}\sum_{j=1}^{m} \Delta_j G(t) \mathbb{1}_{\{G(t)\geq z\}} \mathbb{1}_{\{I_t=j\}}. \end{aligned}$$

By taking the expectation we have that

$$\begin{aligned} \mathbb{E}[R_n] &= \sum_{t=1}^{n}\sum_{j=1}^{m} \Delta_j G(t) \mathbb{1}_{\{G(t)<z\}} \mathbb{P}(\{I_t=j\}) + \\ &+ \sum_{t=1}^{n}\sum_{j=1}^{m} \Delta_j G(t) \mathbb{1}_{\{G(t)\geq z\}} \mathbb{P}(\{I_t=j\}), \end{aligned}$$

which can be rewritten as

$$\begin{aligned} \mathbb{E}[R_n] &= \sum_{t=1}^{n}\sum_{j=1}^{m} \Delta_j G(t) \mathbb{1}_{\{G(t)<z\}} \left[\varepsilon_t \frac{1}{m} + (1-\varepsilon_t)\mathbb{P}(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{i,T_i(t-1)} \ \forall i)\right] \\ &+ \sum_{t=1}^{n}\sum_{j=1}^{m} \Delta_j G(t) \mathbb{1}_{\{G(t)\geq z\}} \mathbb{P}(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{i,T_i(t-1)} \ \forall i). \end{aligned} \tag{32}$$

For the rounds of the algorithm where $G(t) < z$, we are in the standard setting, so for those times, we follow the standard proof of Auer et al. [2002]. For the times that are over the threshold, we need to create a separate bound. Let us now bound the probability of playing the sub-optimal arm j at time t when the greed function is above the threshold z.

$$\begin{aligned} \mathbb{P}(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{i,T_i(t-1)} \ \forall i) &\leq \mathbb{P}(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{*,T_*(t-1)}) & (33)\\ &\leq \mathbb{P}\left(\widehat{X}_{j,T_j(t-1)} \geq \mu_j + \frac{\Delta_j}{2}\right) + \mathbb{P}\left(\widehat{X}_{*,T_*(t-1)} \leq \mu_* - \frac{\Delta_j}{2}\right), & (34) \end{aligned}$$

where the last inequality follows from the fact that

$$\left\{\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{*,T_*(t-1)}\right\} \subset \left(\left\{\widehat{X}_{*,T_*(t-1)} \leq \mu_* - \frac{\Delta_j}{2}\right\} \cup \left\{\widehat{X}_{j,T_j(t-1)} \geq \mu_j + \frac{\Delta_j}{2}\right\}\right). \tag{35}$$

In fact, suppose that there exist an element $\omega \in \left\{\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{*,T_*(t-1)}\right\}$ that does not belong to $\left(\left\{\widehat{X}_{*,T_*(t-1)} \leq \mu_* - \frac{\Delta_j}{2}\right\} \cup \left\{\widehat{X}_{j,T_j(t-1)} \geq \mu_j + \frac{\Delta_j}{2}\right\}\right)$. Then, we would have that

$$\begin{aligned} \omega &\in \left(\left\{\widehat{X}_{*,T_*(t-1)} \leq \mu_* - \frac{\Delta_j}{2}\right\} \cup \left\{\widehat{X}_{j,T_j(t-1)} \geq \mu_j + \frac{\Delta_j}{2}\right\}\right)^C & (36)\\ &= \left\{\widehat{X}_{*,T_*(t-1)} > \mu_* - \frac{\Delta_j}{2}\right\} \cap \left\{\widehat{X}_{j,T_j(t-1)} < \mu_j + \frac{\Delta_j}{2}\right\}, & (37) \end{aligned}$$

but from the intersection of events given in (37) it follows that $\widehat{X}_{*,T_*(t-1)} > \mu_* - \frac{\Delta_j}{2} = \mu_j + \frac{\Delta_j}{2} > \widehat{X}_{j,T_j(t-1)}$ which contradicts $\omega \in \left\{\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{*,T_*(t-1)}\right\}$.

Therefore, all elements of $\left\{\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{*,T_*(t-1)}\right\}$ belong to $\left(\left\{\widehat{X}_{*,T_*(t-1)} \leq \mu_* - \frac{\Delta_j}{2}\right\} \cup \left\{\widehat{X}_{j,T_j(t-1)} \geq \mu_j + \frac{\Delta_j}{2}\right\}\right)$. Let us consider the first term of (34) (the computations for the second term are similar),

$$\begin{aligned}
\mathbb{P}\left(\widehat{X}_{j,T_j(t-1)} \geq \mu_j + \frac{\Delta_j}{2}\right) &= \sum_{s=1}^{t-1} \mathbb{P}\left(T_j(t-1) = s, \widehat{X}_{j,s} \geq \mu_j + \frac{\Delta_j}{2}\right) \\
&= \sum_{s=1}^{t-1} \mathbb{P}\left(T_j(t-1) = s | \widehat{X}_{j,s} \geq \mu_j + \frac{\Delta_j}{2}\right) \mathbb{P}\left(\widehat{X}_{j,s} \geq \mu_j + \frac{\Delta_j}{2}\right) \\
&\leq \sum_{s=1}^{t-1} \mathbb{P}\left(T_j(t-1) = s | \widehat{X}_{j,s} \geq \mu_j + \frac{\Delta_j}{2}\right) e^{-\frac{\Delta_j^2}{2} s}, \qquad (38)
\end{aligned}$$

where in the last inequality we used the Chernoff-Hoeffdings bound. Let us define $T_j^R(t-1)$ as the number of times arm j is played at random (note that $T_j^R(t-1) \leq T_j(t-1)$ and that $T_j^R(t-1) = \sum_{s=1}^{t-1} B_s$ where B_s is a Bernoulli r.v. with parameter ε_s/m), and let us define

$$\lambda_t = \frac{1}{2m} \sum_{s=1}^{\tilde{t}} \varepsilon_s,$$

where $\tilde{t}$ is the number of rounds played under the threshold z up to time t. Then,

$$\begin{aligned}
(38) &\leq \sum_{s=1}^{\lfloor \lambda_t \rfloor} \mathbb{P}\left(T_j(t-1) = s | \widehat{X}_{j,s} \geq \mu_j + \frac{\Delta_j}{2}\right) + \sum_{s=\lfloor \lambda_t \rfloor + 1}^{t-1} e^{-\frac{\Delta_j^2}{2} s} \\
&\leq \sum_{s=1}^{\lfloor \lambda_t \rfloor} \mathbb{P}\left(T_j(t-1) = s | \widehat{X}_{j,s} \geq \mu_j + \frac{\Delta_j}{2}\right) + \frac{2}{\Delta_j^2} e^{-\frac{\Delta_j^2}{2} \lfloor \lambda_t \rfloor} \\
&\leq \sum_{s=1}^{\lfloor \lambda_t \rfloor} \mathbb{P}\left(T_j^R(t-1) \leq s | \widehat{X}_{j,s} \geq \mu_j + \frac{\Delta_j}{2}\right) + \frac{2}{\Delta_j^2} e^{-\frac{\Delta_j^2}{2} \lfloor \lambda_t \rfloor} \\
&\leq \lfloor \lambda_t \rfloor \mathbb{P}\left(T_j^R(t-1) \leq \lfloor \lambda_t \rfloor\right) + \frac{2}{\Delta_j^2} e^{-\frac{\Delta_j^2}{2} \lfloor \lambda_t \rfloor} \qquad (39)
\end{aligned}$$

where for the first $\lfloor \lambda_t \rfloor$ terms of the sum we upperbounded $e^{-\frac{\Delta_j^2}{2} s}$ by 1, and for the remaining terms we used the fact that $\sum_{\lfloor \lambda_t \rfloor + 1}^{\infty} e^{-ks} \leq \frac{1}{k} e^{-kx}$, where in our case $k = \frac{\Delta_j^2}{2}$. We have that

$$\mathbb{E}[T_j^R(t-1)] = \frac{1}{m} \sum_{s=1}^{\tilde{t}} \varepsilon_s, \quad Var(T_j^R(t-1)) = \sum_{s=1}^{\tilde{t}} \frac{\varepsilon_s}{m}\left(1 - \frac{\varepsilon_s}{m}\right) \leq \frac{1}{m} \sum_{s=1}^{\tilde{t}} \varepsilon_s = \mathbb{E}[T_j^R(t-1)],$$

and, using the Bernstein inequality $\mathbb{P}(S_n \leq \mathbb{E}[S_n] - a) \leq \exp\{-\frac{a^2/2}{\sigma^2 + a/2}\}$ with $S_n = T_j^R(t-1)$ and $a = \frac{1}{2}\mathbb{E}[T_j^R(t-1)]$,

$$\begin{aligned}
\mathbb{P}(T_j^R(t-1) \leq \lfloor \lambda_t \rfloor) &= \mathbb{P}\left(T_j^R(t-1) \leq \mathbb{E}[T_j^R(t-1)] - \frac{1}{2}\mathbb{E}[T_j^R(t-1)]\right) \\
&\leq \exp\left\{-\frac{\frac{1}{8}(\mathbb{E}[T_j^R(t-1)])^2}{\mathbb{E}[T_j^R(t-1)] + \frac{1}{4}\mathbb{E}[T_j^R(t-1)]}\right\} \\
&= \exp\left\{-\frac{4}{5}\frac{1}{8}\mathbb{E}[T_j^R(t-1)]\right\} = \exp\left\{-\frac{1}{5}\lfloor \lambda_t \rfloor\right\}. \qquad (40)
\end{aligned}$$

Now we need a lower bound on $\lfloor \lambda_t \rfloor$. Let us define $n' = \lfloor km \rfloor$, then

$$\begin{aligned}
\lambda_t &= \frac{1}{2m}\sum_{s=1}^{\tilde{t}} \varepsilon_s \\
&= \frac{1}{2m}\sum_{s=1}^{\tilde{t}} \min\left\{1, \frac{km}{s}\right\} \\
&= \frac{1}{2m}\sum_{s=1}^{n'} 1 + \frac{1}{2m}\sum_{s=n'+1}^{\tilde{t}} \frac{km}{s} \\
&= \frac{n'}{2m} + \frac{1}{2m}\left(\sum_{s=1}^{\tilde{t}} \frac{km}{s} - \sum_{s=1}^{n'} \frac{km}{s}\right) \\
&\geq \frac{n'}{2m} + \frac{k}{2}\left(\log(\tilde{t}+1) - (\log(n') + \log(e)\right) \\
&\geq \frac{k}{2}\log\left(\frac{n'}{m}\frac{1}{k}\right) + \frac{k}{2}\log\left(\frac{\tilde{t}}{n'e}\right) \\
&= \frac{k}{2}\log\left(\frac{\tilde{t}}{mke}\right).
\end{aligned} \tag{41}$$

Remark 1. Note that if $\tilde{t}$ (or t in the usual ε-greedy algorithm) was less than n', then we would have $\lambda_t = \tilde{t}/2m$, yielding an exponential decay of the bound on the probability of j being the best arm. To see this, $\tilde{t} < n'$ would imply that, using (39) and (40),

$$(39) \leq \frac{\tilde{t}}{2m}\exp\left\{-\frac{1}{5}\frac{\tilde{t}}{2m}\right\} + \frac{2}{\Delta_j^2}\exp\left\{-\frac{\Delta_j^2}{2}\frac{\tilde{t}}{2m}\right\}.$$

Continuing the proof of Theorem 3.1, we obtain a bound on the first term in (34) as follows. Using (41) combined with (40) in (39), we get that

$$\frac{k}{2}\left(\frac{\tilde{t}}{mke}\right)^{-\frac{k}{10}}\log\left(\frac{\tilde{t}}{mke}\right) + \frac{2}{\Delta_j^2}\left(\frac{\tilde{t}}{mke}\right)^{-\frac{k\Delta_j^2}{4}}. \tag{42}$$

Since the computations for the second term in (34) are similar, a bound on $\mathbb{P}\left(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{i,T_i(t-1)} \;\; \forall i\right)$ is given by

$$\beta_j(\tilde{t}) = k\left(\frac{\tilde{t}}{mke}\right)^{-\frac{k}{10}}\log\left(\frac{\tilde{t}}{mke}\right) + \frac{4}{\Delta_j^2}\left(\frac{\tilde{t}}{mke}\right)^{-\frac{k\Delta_j^2}{4}}. \tag{43}$$

We have now an upper bound for $\mathbb{P}(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{i,T_i(t-1)} \;\; \forall i)$. We can use this to easily bound $\mathbb{P}(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{i,T_i(t-1)} \;\; \forall i)$ in (32) which yields the following bound on the mean regret at time n:

$$\begin{aligned}
\mathbb{E}[R_n] &\leq \sum_{j=1}^{m} G(j)\Delta_j \\
&+ \sum_{t=m+1}^{n} G(t)\mathbb{1}_{\{G(t)<z\}} \sum_{j:\mu_j<\mu_*} \Delta_j\left(\varepsilon_t\frac{1}{m} + (1-\varepsilon_t)\beta_j(\tilde{t})\right) \\
&+ \sum_{t=m+1}^{n} G(t)\mathbb{1}_{\{G(t)\geq z\}} \sum_{j:\mu_j<\mu_*} \Delta_j\beta_j(\tilde{t}),
\end{aligned}$$

This, combined with the bound $\beta_j(\tilde{t})$ above, proves the theorem.

B Logarithmic bound for Soft-ε-greedy algorithm

At each round t, arm j is played with probability

$$\frac{\varepsilon_t}{m} + (1-\varepsilon_t)\mathbb{P}\left(\widehat{X}_j \geq \widehat{X}_i \; \forall i\right),$$

where $\varepsilon_t = \min\left\{\psi(t), \frac{km}{t}\right\}$ and

$$\psi(t) = \frac{\log\left(1 + \frac{1}{G(t)}\right)}{\log\left(1 + \frac{1}{\min_{s\in\{m+1,\cdots,n\}} G(s)}\right)}. \tag{44}$$

Recall that $\gamma = \min_{1\leq t\leq n} \psi(t)$.
Let us bound the probability $\mathbb{P}(\{I_t = j\})$ of playing the sub-optimal arm j at time t. We have that

$$\begin{aligned}
\mathbb{P}\left(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{i,T_i(t-1)}\ \forall i\right) &\leq \mathbb{P}\left(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{*,T_*(t-1)}\right) \\
&\leq \mathbb{P}\left(\widehat{X}_{j,T_j(t-1)} \geq \mu_j + \frac{\Delta_j}{2}\right) + \mathbb{P}\left(\widehat{X}_{j,T_j(t-1)} \leq \mu_* + \frac{\Delta_j}{2}\right). \qquad (45)
\end{aligned}$$

For the bound on the two addends in (45), we have identical steps to the proof for Theorem 1, and thus

$$\begin{aligned}
\mathbb{P}\left(\widehat{X}_{j,T_j(t-1)} \geq \mu_j + \frac{\Delta_j}{2}\right) &\leq \lfloor\lambda_t\rfloor \mathbb{P}\left(T_j^R(t-1) \leq \lfloor\lambda_t\rfloor\right) + \frac{2}{\Delta_j^2} e^{-\frac{\Delta_j^2}{2}\lfloor\lambda_t\rfloor} \qquad (46) \\
\mathbb{P}\left(\widehat{X}_{j,T_j(t-1)} \leq \mu_* + \frac{\Delta_j}{2}\right) &\leq \lfloor\lambda_t\rfloor \mathbb{P}\left(T_j^R(t-1) \leq \lfloor\lambda_t\rfloor\right) + \frac{2}{\Delta_j^2} e^{-\frac{\Delta_j^2}{2}\lfloor\lambda_t\rfloor} \qquad (47)
\end{aligned}$$

and we again have

$$\mathbb{P}\left(T_j^R(t-1) \leq \lfloor\lambda_t\rfloor\right) \leq \exp\left\{-\frac{1}{5}\lfloor\lambda_t\rfloor\right\}. \tag{48}$$

Now we need a lower bound on $\lfloor\lambda_t\rfloor$. Let $t > w$ where $w = \min\{1, \cdots, n\}$ such that $\frac{cm}{d^2 w} < \gamma$. Then,

$$\begin{aligned}
\lambda_t &= \frac{1}{2m}\sum_{s=1}^{t} \varepsilon_s \\
&= \frac{1}{2m}\sum_{s=1}^{t} \min\left\{\psi(s), \frac{km}{s}\right\} \\
&\geq \frac{1}{2m}\sum_{s=1}^{w} \gamma + \frac{1}{2m}\left(\sum_{s=1}^{t}\frac{km}{s} - \sum_{s=1}^{w}\frac{km}{s}\right) \\
&\geq \frac{w\gamma}{2m} + \frac{k}{2}\left(\log(t+1) - (\log(w) + \log(e)\right) \\
&\geq \frac{k}{2}\log\left(\frac{w\gamma}{m}\frac{1}{k}\right) + \log\left(\frac{t}{we}\right) \\
&= \frac{k}{2}\log\left(\frac{\gamma t}{mke}\right). \qquad (49)
\end{aligned}$$

Using (49) in (48), combined with (46) and (47), from (45) the bound on $\mathbb{P}\left(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{i,T_i(t-1)}\ \forall i\right)$ is given by

$$\beta_j^S(t) = k\log\left(\frac{\gamma t}{mke}\right)\left(\frac{\gamma t}{mke}\right)^{-\frac{k}{10}} + \frac{4}{\Delta_j^2}\left(\frac{\gamma t}{mke}\right)^{-\frac{k\Delta_j^2}{4}}.$$

Since the mean regret is given by

$$\mathbb{E}[R_n] = \sum_{t=1}^{n}\sum_{j=1}^{m} \Delta_j G(t)\mathbb{P}\left(\{I_t = j\}\right), \tag{50}$$

the bound on the mean regret at time n is given by

$$\begin{aligned}
\mathbb{E}[R_n] &\leq \sum_{j=1}^{m} G(j)\Delta_j \\
&+ \sum_{t=m+1}^{n} G(t)\sum_{j=1}^{m}\Delta_j\left(\varepsilon_t\frac{1}{m} + (1-\varepsilon_t)\beta_j^S(t)\right).
\end{aligned}$$

C The regret bound of the soft UCB algorithm

The regret at round n is given by

$$\begin{aligned} R_n &= \sum_{j=1}^{m} G(j)\Delta_j + \sum_{t=m+1}^{n}\sum_{j=1}^{m} \Delta_j G(t)\mathbb{1}_{\{I_t=j\}} \\ &\le \sum_{j=1}^{m} G(j)\Delta_j + \left(\max_{t\in\{m+1,\cdots,n\}} G(t)\right)\sum_{j=1}^{m}\Delta_j \sum_{t=m+1}^{n}\mathbb{1}_{\{I_t=j\}}. \end{aligned}$$

The expected regret $\mathbb{E}[R_n]$ at round n is bounded by

$$\mathbb{E}[R_n] \le \sum_{j=1}^{m} G(j)\Delta_j + \left(\max_{t\in\{m+1,\cdots,n\}} G(t)\right)\sum_{j=1}^{m}\Delta_j \mathbb{E}[T_j(n)]. \tag{51}$$

where $T_j(n) = \sum_{t=1}^{n} \mathbb{1}_{\{I_t=j\}}$ is the number of times the sub-optimal arm j has been chosen up to round n. Recall from (1) that

$$\widehat{X}_j = \frac{1}{T_j(t-1)}\sum_{s=1}^{T_j(t-1)} X_j(s). \tag{52}$$

From the Chernoff-Hoeffding Inequality we have that

$$\mathbb{P}\left(\frac{1}{T_j(t-1)}\sum_{i=1}^{T_j(t-1)} X_{j,i} - \mu_j \le -\varepsilon\right) \le \exp\{-2T_j(t-1)\varepsilon^2\},$$

and

$$\mathbb{P}\left(\frac{1}{T_j(t-1)}\sum_{i=1}^{T_j(t-1)} X_{j,i} - \mu_j \ge \varepsilon\right) \le \exp\{-2T_j(t-1)\varepsilon^2\}. \tag{53}$$

Let us define the following function:

$$\xi(t) = \left(1 + \frac{t}{G(t)}\right),$$

by selecting $\varepsilon = \sqrt{\frac{2\log\xi(t)}{T_j(t-1)}}$ we have

$$\mathbb{P}\left(\widehat{X}_j + \sqrt{\frac{2\log\xi(t)}{T_j(t-1)}} \le \mu_j\right) \le \xi(t)^{-4}, \tag{54}$$

and

$$\mathbb{P}\left(\widehat{X}_j - \sqrt{\frac{2\log\xi(t)}{T_j(t-1)}} \ge \mu_j\right) \le \xi(t)^{-4}. \tag{55}$$

Equivalently, we may write for every j

$$\mu_j - \sqrt{\frac{2\log\xi(t)}{T_j(t-1)}} \le \widehat{X}_j \quad \text{with probability at least } 1-\xi(t)^{-4}, \tag{56}$$

$$\mu_j + \sqrt{\frac{2\log\xi(t)}{T_j(t-1)}} \ge \widehat{X}_j \quad \text{with probability at least } 1-\xi(t)^{-4}. \tag{57}$$

If we choose arm j at round t (i.e., the event $\{I_t = j\}$ occurs) we have that

$$\widehat{X}_j + \sqrt{\frac{2\log\xi(t)}{T_j(t-1)}} \ge \widehat{X}_* + \sqrt{\frac{2\log\xi(t)}{T_*(t-1)}}. \tag{58}$$

Let us use (57) to upper bound the LHS and (56) to lower bound the RHS of (58), then we get with probability at least $1 - 2\xi(t)^{-4}$

$$\mu_j + 2\sqrt{\frac{2\,\log\xi(t)}{T_j(t-1)}} \geq \mu_*,$$

from which we get with probability at least $1 - 2\xi(t)^{-4}$

$$T_j(t-1) \leq \frac{8}{\Delta_j^2}\log\xi(t). \tag{59}$$

In order to emphasize the dependence of $\widehat{X}_j$ from $T_j(t-1)$ we will sometimes write $\widehat{X}_{j,T_j(t-1)}$. In the following, notice that in (61) the summation starts from $m+1$ because in the first m initialization rounds each arm is played once. Moreover, step (62) follows from (61) by assuming that arm j has already been played u times. By using (58) we get (63), then, for each t,

$$\left\{\widehat{X}_{j,T_j(t-1)} + \sqrt{\frac{2\log\xi(t)}{T_j(t-1)}} \geq \widehat{X}_{*,T_*(t-1)} + \sqrt{\frac{2\log\xi(t)}{T_*(t-1)}}, T_j(t-1) \geq u\right\} \subset$$
$$\left\{\max_{s_j\in\{u,\dots,T_j(t-1)\}} \widehat{X}_{j,s_j} + \sqrt{\frac{2\log\xi(t)}{s_j}} \geq \min_{s_*\in\{1,\dots,T_*(t-1)\}} \widehat{X}_{*,s_*} + \sqrt{\frac{2\log\xi(t)}{s_*}}\right\} \tag{60}$$

which justifies (64). We also have that (60) is included in

$$\bigcup_{s_*=1}^{T_*(t-1)}\bigcup_{s_j=u}^{T_j(t-1)}\left\{\widehat{X}_{j,s_j} + \sqrt{\frac{2\log\xi(t)}{s_j}} \geq \widehat{X}_{*,s_*} + \sqrt{\frac{2\log\xi(t)}{s_*}}\right\}.$$

Thus, for any integer u, we may write

$$\begin{aligned}
T_j(n) &= 1 + \sum_{t=m+1}^{n} \mathbb{1}\{I_t = j\} && (61)\\
&= u + \sum_{t=m+1}^{n} \mathbb{1}\{I_t = j, T_j(t-1) \geq u\} && (62)\\
&= u + \sum_{t=m+1}^{n} \mathbb{1}\left\{\widehat{X}_{j,T_j(t-1)} + \sqrt{\frac{2\log\xi(t)}{T_j(t-1)}} \geq \widehat{X}_{*,T_*(t-1)} + \sqrt{\frac{2\log\xi(t)}{T_*(t-1)}}, T_j(t-1) \geq u\right\} && (63)\\
&\leq u + \sum_{t=m+1}^{n} \mathbb{1}\left\{\max_{s_j\in\{u,\dots,T_j(t-1)\}} \widehat{X}_{j,s_j} + \sqrt{\frac{2\log\xi(t)}{s_j}} \geq \min_{s_*\in\{1,\dots,T_*(t-1)\}} \widehat{X}_{*,s_*} + \sqrt{\frac{2\log\xi(t)}{s_*}}\right\} && (64)\\
&\leq u + \sum_{t=m+1}^{n}\sum_{s_*=1}^{T_*(t-1)}\sum_{s_j=u}^{T_j(t-1)} \mathbb{1}\left\{\widehat{X}_{j,s_j} + \sqrt{\frac{2\log\xi(t)}{s_j}} \geq \widehat{X}_{*,s_*} + \sqrt{\frac{2\log\xi(t)}{s_*}}\right\}. && (65)
\end{aligned}$$

When

$$\mathbb{1}\left\{\widehat{X}_j + \sqrt{\frac{2\,\log\xi(t)}{T_j(t-1)}} \geq \widehat{X}_* + \sqrt{\frac{2\,\log\xi(t)}{T_*(t-1)}}\right\} \tag{66}$$

is equal to one, at least one of the following has to be true:

$$\begin{aligned}
\widehat{X}_* &\leq \mu_* - \sqrt{\frac{2\,\log\xi(t)}{T_*(t-1)}}; && (67)\\
\widehat{X}_j &\geq \mu_j + \sqrt{\frac{2\,\log\xi(t)}{T_j(t-1)}}; && (68)\\
\mu_* &< \mu_j + 2\sqrt{\frac{2\,\log\xi(t)}{T_j(t-1)}}. && (69)
\end{aligned}$$

(In fact, suppose none of them hold simultaneously. Then from (67) we would have that $\widehat{X}_* > \mu_* - \sqrt{\frac{2\log\xi(t)}{T_*(t-1)}}$; then, by applying (69) (with opposite verse since we are assuming it does not hold) we get $\widehat{X}_* > \mu_j + 2\sqrt{\frac{2\log\xi(t)}{T_j(t-1)}} - \sqrt{\frac{2\log\xi(t)}{T_*(t-1)}}$ and then from (68) (again, with opposite verse) follows that $\widehat{X}_* > \widehat{X}_j + \sqrt{\frac{2\log\xi(t)}{T_j(t-1)}} - \sqrt{\frac{2\log\xi(t)}{T_*(t-1)}}$ which is in contradiction with (66).) Now, if we set $u = \left\lceil \frac{8}{\Delta_j^2} \log\left(\max_{t\in\{m+1,\cdots,n\}} \xi(t)\right)\right\rceil$, for $T_j(t-1) \geq u$,

$$\begin{aligned}
& \mu_* - \mu_j - 2\sqrt{\frac{2\log\xi(t)}{T_j(t-1)}} \\
\geq \; & \mu_* - \mu_j - 2\sqrt{\frac{2\log\xi(t)}{u}} \\
= \; & \mu_* - \mu_j - \Delta_j \sqrt{\frac{\log\xi(t)}{\left\lceil \log\left(\max_{t\in\{m+1,\cdots,n\}} \xi(t)\right)\right\rceil}} \\
\geq \; & \mu_* - \mu_j - \Delta_j = 0,
\end{aligned}$$

therefore, with this choice of u, (69) can not hold.

Thus, using (65), we have that

$$\begin{aligned}
T_j(n) \;\leq\; & \left\lceil \frac{8}{\Delta_j^2} \log\left(\max_{t\in\{m+1,\cdots,n\}} \xi(t)\right)\right\rceil \\
& + \sum_{t=m+1}^{n} \sum_{s_*=1}^{T_*(t-1)} \sum_{s_j=u}^{T_j(t-1)} \mathbb{1}\left\{\widehat{X}_{*,s_*} \leq \mu_* - \sqrt{\frac{2\log\xi(t)}{s_*}}\right\} \\
& + \sum_{t=m+1}^{n} \sum_{s_*=1}^{T_*(t-1)} \sum_{s_j=u}^{T_j(t-1)} \mathbb{1}\left\{\widehat{X}_{j,s_j} \geq \mu_j + \sqrt{\frac{2\log\xi(t)}{s_j}}\right\}
\end{aligned}$$

and by taking expectation,

$$\begin{aligned}
\mathbb{E}[T_j(n)] \;\leq\; & \left\lceil \frac{8}{\Delta_j^2} \log\left(\max_{t\in\{m+1,\cdots,n\}} \xi(t)\right)\right\rceil \\
& + \sum_{t=m+1}^{n} \sum_{s_*=1}^{T_*(t-1)} \sum_{s_j=u}^{T_j(t-1)} \mathbb{P}\left\{\widehat{X}_{*,s_*} \leq \mu_* - \sqrt{\frac{2\log\xi(t)}{s_*}}\right\} \\
& + \sum_{t=m+1}^{n} \sum_{s_*=1}^{T_*(t-1)} \sum_{s_j=u}^{T_j(t-1)} \mathbb{P}\left\{\widehat{X}_{j,s_j} \geq \mu_j + \sqrt{\frac{2\log\xi(t)}{s_j}}\right\} \\
\leq \; & \frac{8}{\Delta_j^2} \log\left(\max_{t\in\{m+1,\cdots,n\}} \xi(t)\right) + 1 + 2\sum_{t=m+1}^{n} \xi(t)^{-4}(t-1-m)^2.
\end{aligned}$$

where in the last step we upperbound $T_*(t-1)$ and $T_j(t-1)$ by $(t-1-m)$ (cases where we have only played the best arm or arm j). Therefore, by using (51)

$$\begin{aligned}
\mathbb{E}[R_n] \;\leq\; & \sum_{j=1}^{m} G(j)\Delta_j \\
& + \max_{t\in\{m+1,\cdots,n\}} G(t) \left(\sum_{j:\mu_j<\mu_*} \frac{8}{\Delta_j} \log\left(\max_{t\in\{m+1,\cdots,n\}} \xi(t)\right) + \sum_{j=1}^{m} \Delta_j \left[1 + \sum_{t=m+1}^{n} 2\,(\xi(t))^{-4}\,(t-1-m)^2\right]\right).
\end{aligned}$$

D Regret bound proof for regulating greed with arm pool size

The regret at round n is given by

$$R_n = \sum_{t=1}^{n}\sum_{j=1}^{m} \Delta_j G(t) \mathbb{1}_{\{I_t=j\}}, \tag{70}$$

where $G(t)$ is the greed function evaluated at time t, $\mathbb{1}_{\{I_t=j\}}$ is an indicator function equal to 1 if arm j is played at time t (otherwise its value is 0) and $\Delta_j = \mu^* - \mu_j$ is the difference between the mean of the best arm reward distribution and the mean of the j's arm reward distribution. Similarly, $\Delta_j^{*-m_t} = \mu^{*-m_t} - \mu_j$ is the difference between the mean reward of the m_tth-best arm and the mean reward of arm j (with this definition we can also write $\Delta_j = \Delta_j^{*-1}$ and $\mu^* = \mu^{*-1}$). By taking the expectation of (70) we get

$$\mathbb{E}[R_n] = \sum_{t=1}^{n} \frac{G(t)}{m_t} \sum_{j=1}^{m} \Delta_j \mathbb{P}\left(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{i,T_i(t-1)} \quad \text{for at least } \ m - m_t \ \text{ indexes } \ i\right), \tag{71}$$

where m_t is the size of the pool of arms at time t, defined by $m_t = \min\left(m, \max\left(1, \frac{cm}{tG(t)}\right)\right)$. We have that

$$\begin{aligned}
&\mathbb{P}\left(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{i,T_i(t-1)} \quad \text{for at least } \ m - m_t \ \text{ indexes } \ i\right) \\
&\leq \mathbb{P}\left(\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{*-m_t,T_{*-m_t}(t-1)}\right) \\
&\leq \mathbb{P}\left(\widehat{X}_{j,T_j(t-1)} \geq \mu_j + \frac{\Delta_j^{*-m_t}}{2}\right) + \mathbb{P}\left(\widehat{X}_{*,T_*(t-1)} \leq \mu^{*-m_t} - \frac{\Delta_j^{*-m_t}}{2}\right)
\end{aligned} \tag{72}$$

the last inequality follows from the fact that

$$\left\{\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{*-m_t,T_{*-m_t}(t-1)}\right\} \subset \left(\left\{\widehat{X}_{j,T_j(t-1)} \geq \mu_j + \frac{\Delta_j^{*-m_t}}{2}\right\} \cup \left\{\widehat{X}_{*,T_*(t-1)} \leq \mu^{*-m_t} - \frac{\Delta_j^{*-m_t}}{2}\right\}\right). \tag{73}$$

In fact, suppose that the inclusion (73) does not hold. Then we would have that

$$\begin{aligned}
\left\{\widehat{X}_{j,T_j(t-1)} \geq \widehat{X}_{*-m_t,T_{*-m_t}(t-1)}\right\} \quad &\subset \quad \left(\left\{\widehat{X}_{*,T_*(t-1)} \leq \mu^{*-m_t} - \frac{\Delta_j^{*-m_t}}{2}\right\} \cup \left\{\widehat{X}_{j,T_j(t-1)} \geq \mu_j + \frac{\Delta_j^{*-m_t}}{2}\right\}\right)^C & (74) \\
&= \quad \left\{\widehat{X}_{*,T_*(t-1)} > \mu^{*-m_t} - \frac{\Delta_j^{*-m_t}}{2}\right\} \cap \left\{\widehat{X}_{j,T_j(t-1)} < \mu_j + \frac{\Delta_j^{*-m_t}}{2}\right\}, & (75)
\end{aligned}$$

but from the intersection of events given in (75) it follows that $\widehat{X}_{*-m_t,T_{*-m_t}(t-1)} > \mu^{*-m_t} - \frac{\Delta_j^{*-m_t} j}{2} > \mu_j - \frac{\Delta_j^{*-m_t}}{2} > \widehat{X}_{j,T_j(t-1)}$ which contradicts (74). Let us consider the first term of (72) (the computations for the second term are similar),

$$\begin{aligned}
\mathbb{P}\left(\widehat{X}_{j,T_j(t-1)} \geq \mu_j + \frac{\Delta_j^{*-m_t}}{2}\right) \quad &= \quad \sum_{s=1}^{t-1} \mathbb{P}\left(T_j(t-1) = s, \widehat{X}_{j,s} \geq \mu_j + \frac{\Delta_j^{*-m_t}}{2}\right) \\
&= \quad \sum_{s=1}^{t-1} \mathbb{P}\left(T_j(t-1) = s \,\middle|\, \widehat{X}_{j,s} \geq \mu_j + \frac{\Delta_j^{*-m_t}}{2}\right) \mathbb{P}\left(\widehat{X}_{j,s} \geq \mu_j + \frac{\Delta_j^{*-m_t}}{2}\right) \\
&\leq \quad \sum_{s=1}^{t-1} \mathbb{P}\left(T_j(t-1) = s \,\middle|\, \widehat{X}_{j,s} \geq \mu_j + \frac{\Delta_j^{*-m_t}}{2}\right) e^{-\frac{s(\Delta_j^{*-m_t})^2}{2}},
\end{aligned} \tag{76}$$

where in the last inequality we used the Chernoff-Hoeffdings bound. Let us define $T_j^R(t-1)$ as the number of times arm j is played at random when the pool size is full before round t starts, and let us define

$$\lambda_t = \frac{1}{2m} \sum_{s=1}^{t} \mathbb{1}\left\{\min\left(m, \max\left(1, \frac{cm}{sG(s)}\right)\right) = m\right\}.$$

Then,

$$\begin{aligned}
(38) \quad &\le \sum_{s=1}^{\lfloor \lambda_t \rfloor} \mathbb{P}\left(T_j(t-1) = s | \widehat{X}_{j,s} \ge \mu_j + \frac{\Delta_j^{*-m_t}}{2} \right) + \sum_{s=\lfloor \lambda_t \rfloor + 1}^{t-1} e^{-\frac{(\Delta_j^{*-m_t})^2}{2} s} \\
&\le \sum_{s=1}^{\lfloor \lambda_t \rfloor} \mathbb{P}\left(T_j(t-1) = s | \widehat{X}_{j,s} \ge \mu_j + \frac{\Delta_j^{*-m_t}}{2} \right) + \frac{2}{\Delta_j^2} e^{-\frac{(\Delta_j^{*-m_t})^2}{2} \lfloor \lambda_t \rfloor} \\
&\le \sum_{s=1}^{\lfloor \lambda_t \rfloor} \mathbb{P}\left(T_j^R(t-1) \le s | \widehat{X}_{j,s} \ge \mu_j + \frac{\Delta_j^{*-m_t}}{2} \right) + \frac{2}{\Delta_j^2} e^{-\frac{(\Delta_j^{*-m_t})^2}{2} \lfloor \lambda_t \rfloor} \\
&\le \lfloor \lambda_t \rfloor \mathbb{P}\left(T_j^R(t-1) \le \lambda_t \right) + \frac{2}{(\Delta_j^{*-m_t})^2} e^{-\frac{(\Delta_j^{*-m_t})^2}{2} \lfloor \lambda_t \rfloor} \qquad (77)
\end{aligned}$$

where for the first $\lfloor \lambda_t \rfloor$ terms of the sum we upper-bounded $e^{-\frac{\Delta_j^2}{2} s}$ by 1, and for the remaining terms we used the fact that $\sum_{\lfloor \lambda_t \rfloor + 1}^{\infty} e^{-ks} \le \frac{1}{k} e^{-kx}$, where in our case $k = \frac{\Delta_j^2}{2}$. Since $T_j^R(t-1)$ is a sum of $\lambda = \sum_{s=1}^{t} \mathbb{1}\{m_s = m\}$ independent Bernoulli r.v. with parameter $1/m_s$, we have that

$$\begin{aligned}
\mathbb{E}[T_j^R(t-1)] &= \frac{1}{m} \sum_{s=1}^{t} \mathbb{1}\left\{ \min\left(m, \max\left(1, \frac{cm}{sG(s)} \right) \right) = m \right\} = 2\lambda_t, \\
Var(T_j^R(t-1)) &= \frac{1}{m}\left(1 - \frac{1}{m} \right) \sum_{s=1}^{t} \mathbb{1}\left\{ \min\left(m, \max\left(1, \frac{cm}{sG(s)} \right) \right) = m \right\} \le \mathbb{E}[T_j^R(t-1)],
\end{aligned}$$

and, using the Bernstein inequality $\mathbb{P}(S_n \le \mathbb{E}[S_n] - a) \le \exp\{-\frac{a^2/2}{\sigma^2 + a/2}\}$ with $S_n = T_j^R(t-1)$ and $a = \frac{1}{2}\mathbb{E}[T_j^R(t-1)]$,

$$\begin{aligned}
\mathbb{P}(T_j^R(t-1) \le \lambda_t) &= \mathbb{P}\left(T_j^R(t-1) \le \mathbb{E}[T_j^R(t-1)] - \frac{1}{2}\mathbb{E}[T_j^R(t-1)] \right) \\
&\le \exp\left\{ -\frac{\frac{1}{8}(\mathbb{E}[T_j^R(t-1)])^2}{\mathbb{E}[T_j^R(t-1)] + \frac{1}{4}\mathbb{E}[T_j^R(t-1)]} \right\} \\
&= \exp\left\{ -\frac{4}{5}\frac{1}{8}\mathbb{E}[T_j^R(t-1)] \right\} = \exp\left\{ -\frac{1}{5}\lambda_t \right\}. \qquad (78)
\end{aligned}$$

In order to bound (78) we need $\lambda_t \ge \gamma \log(t)$ with $\gamma > 5$ so that $\mathbb{P}(T_j^R(t-1) \le \lambda_t) < t^{-\frac{\gamma}{5}}$ (if $G(t)$ does not satisfy the requirement that $\lambda_t \ge \gamma \log(t)$ it is easy to construct $G'(t) = (c-1)/t$ for $t \in \{t, t+1, \cdots, t+2m\}$ if $\gamma \log(t) > \lceil \gamma \log(t-1) \rceil$, otherwise $G'(t) = G(t)$.)

Then, (77) is bounded by

$$\beta_j(t) = \gamma \log(t)(t)^{-\gamma/5} + \frac{2}{(\Delta_j^{*-m_t})^2} t^{-\frac{\gamma(\Delta_j^{*-m_t})^2}{2}} \qquad (79)$$

The computations for the second term in (76) are similar, therefore

$$\mathbb{E}[R_n] \le \sum_{t=1}^{n} \sum_{j=1}^{m} \Delta_j G(t) \frac{2}{m_t} \beta_j, \qquad (80)$$

E Notation summary

- m: number of arms;
- n: number of rounds;
- $G : \{1, \cdots, n\} \to \mathbb{R}^+$: known multiplier function;
- $X_j(t)$: unscaled random reward for playing arm j;
- $X_j(t)G(t)$: actual reward;
- μ_*: mean reward of the optimal arm ($\mu_* = \max_{1 \le j \le m} \mu_j$);
- Δ_j: difference between the mean reward of the optimal arm and the mean reward of arm j ($\Delta_j = \mu_* - \mu_j$);
- $\hat{X}_j$: current estimate of μ_j;
- I_t: arm played at turn t;
- $T_j(t-1)$: number of times arm j has been played before round t starts;
- z threshold (used in Algorithm 2 and Algorithm 4);
- $\tilde{t}$: number of rounds under the threshold z up to time t;
- k: a constant greater than 10 such that $k > \frac{4}{\min_j \Delta_j}$ in Algorithm 2 and Algorithm 3;
- c: a constant greater than 10 in Algorithm 7;
- d: a constant such that $d < \min_j \Delta_j$ and $0 < d < 1$ in Algorithm 7;
- ε_t: probability of exploration at turn t (used in Algorithm 2 and Algorithm 3);
- $\beta_j(\tilde{t})$: upper bound on the probability of considering suboptimal arm j being the best arm at round $\tilde{t}$ when using Algorithm 2;
- $\beta_j^{\text{old}}(t)$: upper bound on the probability of considering suboptimal arm j being the best arm at round t when using Algorithm 7;
- $\beta_j^S(t)$: upper bound on the probability of considering suboptimal arm j being the best arm at round t when using Algorithm 3;
- $\psi(t)$: smoothing function used to define the probabilities of exploration ε_t in Algorithm 3 (see Figure 2);
- γ: lowest value of $\psi(t)$ ($\gamma = \min_{s \in \{m+1, \cdots, n\}} \psi(s)$);
- n': particular time defined as km in the comparison between Algorithm 3 and Algorithm 7 in Section 3.4;
- w: first round when $\frac{km}{s}$ is less than γ ($w = \operatorname{argmin} f(s)$, subject to $f(s) < \gamma$, where $f(s) = \frac{km}{s}$) in the comparison between Algorithm 3 and Algorithm 7 in Section 3.4;
- B set of rounds when the "high reward" zone is entered in Algorithm 4 ($B = \{t : G(t-1) < z, G(t) > z\}$);
- $Y_k = \{t : t \ge y_k, G(t) > z, t < y_{k+1}\}$: set of rounds in the high-reward period entered at time y_k ($k \in \{1, \cdots, |B|\}$) in Algorithm 4;
- $\Lambda_k = \max_{t \in Y_k} G(t)$: highest value of $G(t)$ on Y_k in Algorithm 4;
- $R_k = \Lambda_k |Y_k|$: the maximum regret of the kth high reward zone in Algorithm 4;
- $\xi(t)$: smoothing function used to define the decision rule in Algorithm 5;
- e: Euler's number;
- R_n: total regret at round n.

APPENDIX B Multi-armed Bandit Problem

Multi-Armed Bandit Problem

Kamesh Munagala

August 11, 2017

Abstract

In this work, we mostly study the budgeted version of sequential decision making problem when each decision generates a reward and incurs a cost. Particularly, we consider the multi-armed bandit problem which has limited feedback. There are two extreme cases for this problem: stochastic and adversarial. Firstly, we consider the stochastic version of the problem in which there is a joint distribution for the cost and reward of each arm at each time step. We compare different algorithms based on some simulations. Then, an impossibility result is provided for the fully adversarial setting in which the reward and cost of each arm at each time step is determined by an adversary. A new input model named leaky bucket is offered to model a weaker version of adversary. We propose a simple policy for a special case of this new model. Then, we consider the Markovian input model and propose a new policy named Recency. We compare this algorithm and another existing policy for the stochastic version of the problem by some experiments. Finally, we consider the contextual bandit with linear payoffs (there is no cost associated with a decision) and try to generalize an existing algorithm for the case in which there is an unknown parameter for the problem to the case when there is one unknown parameter for each arm.

1 Introduction

Multi-armed bandit (MAB) problems are used to model exploration-exploitation trade-offs that are inherent in many sequential decision making problems. Different versions of this problem have been studied, and different algorithms have been devised to solve them. These algorithms have a wide range of applications, including in medical trials, online advertising, recommender systems, and scheduling.

The basic idea is that we are given a set of options (or arms), and each arm has an unknown associated reward with it. At each time step, we have to decide which arm to play so as to maximize our total reward. In MAB problems the feedback is limited which means after playing an arm in a time step, the player can only see the reward of that arm in the last step and does not obtain any information about the other options. The full feedback version of the problem called expert problem has been also studied and some of the MAB algorithms (e.g. EXP3) leverage the existing algorithms for expert problem to solve the MAB problem.

This problem can be adapted to several different settings. We mainly consider the MAB problem in the presence of cost that each arm incurs a cost, as well as produces a reward at each time step (note, the value of the cost and reward at each time step may be correlated). There is one budget, and we can continue to choose an arm at each step until the total amount of cost exceeds the budget. We define our benchmark as the single best arm (playing that arm at each time step). The regret of an algorithm is defined as the difference between the total reward of our benchmark and the total reward of the algorithm. The objective is to design an algorithm that minimizes regret. This problem (and a more general version of it) has been already studied in the stochastic environment in which there is a joint distribution for the reward and cost of each arm, and the reward and cost for that arm is an independent sample from its unknown distribution at each time step. In the first part of this work, we also consider this case and compare different algorithms by simulations.

Then, we investigate whether similar results can be obtained for a non-stochastic environment. Particularly, we consider adversarial model of the problem in which the reward and cost of each arm at each time step is chosen by an adversary. For this setting, we prove some negative results showing the adversary is too powerful to compete against. Then, we define a weaker adversarial model named leaky bucket input model and propose a simple algorithm for this new version of the problem. Then, we consider a Markovian input model, in which for each arm there exists an underlying Markov chain with few states. We propose a new algorithm named Recency to adapt to this changing environment. We implement this algorithm and compare the results in some sample input with the UCB-Simplex which was originally designed for stochastic input model in [9].

Finally, we consider the contextual bandit with linear payoffs problem. In this version of the problem, we assume that the number of rounds that we need to pull an arm is given and each arm only produces a reward at each time step. In this section, we try to generalize the Thompson Sampling algorithm in [2] to another version of the problem.

2 Problem Statement

There are m different arms and each arm i produces reward $r_i^t \in [0,1]$ and incurs cost $c_i^t \in [0,1]$ at time step t. There are different models for the reward and cost of the arms. The first one is stochastic model which has received a lot of attention: in this model there exists a fixed unknown joint distribution for the reward and cost of each arm and choosing an arm at a time step results in an independent sample from its distribution determining the reward and cost of that choice. We mention some of the existing results for this model in the next section. The other model is the adversarial model in which the reward and cost of each arm at each time step are determined by an adversary. In this case, the history of the rewards and costs of an arm in the previous rounds tells nothing about these values in the current time step.

The budget denoted by B determines the stopping time of the game. The player continues to choose an arm at each time step until the total amount of costs exceeds the budget. Let $p(t)$ denote the chosen arm at time t by policy p, the stopping time of the policy p denoted by $s(p)$ should satisfy the following:

$$\sum_{t=1}^{s(p)} c_{p(t)}^t \leq B \text{ and } \sum_{t=1}^{s(p)+1} c_{p(t)}^t > B$$

The regret of an algorithm is defined as the difference between the total reward of a benchmark and the total reward of the algorithm. In this work, we consider the best single arm policy as the benchmark. Let p_i denote the single arm policy that always play arm i, then for any policy p the regret is defined as follows:

$$R(p) = \sum_{t=1}^{s(p)} r_{p(t)}^t - \max_{i \in \{1,2,\dots m\}} \sum_{t=1}^{s(p_i)} r_i^t$$

3 Related Work

Ding et al. in [7], study the multi-armed bandit problem with budget constraint and variable costs for the first time. In their setting, there exists one distribution for the reward of each arm.

In addition, there exists a multinomial distribution for the cost of each arm. They assume that the cost is between 0 and 1 and it is always a multiple of the unit cost. The reward and cost of arm i at each time step is an iid sample from the corresponding distributions. They propose two algorithms for this problem with regret bound of $O(\ln B)$.

Badanidiyuru et al. in [5] have tackled a more general *bandit with knapsacks* (BWK) problem. In this setting, there are d different resources, each one with its own budget. The process of choosing an arm continues until at least one of the resources has been fully consumed. They particularly looked at a stochastic version of the problem, where the reward and costs for each arm is an independent sample from an unknown joint distribution. In this general setting, the best single arm may not be a strong benchmark. For example if there exist 2 different resources and 2 arms where arm 1 only consumes the first resource and arm 2 only consumes the second resource, an optimal policy should play both of the arms. They consider the stronger benchmark which is the optimal dynamic policy when the joint distribution is known upfront.

They propose two algorithms, with regret $\widetilde{O}(\sqrt{mOPT} + OPT\sqrt{\frac{m}{B}})$. Here, m is the number of arms, B is the minimum of all the budgets, and OPT is the expected reward of the optimal policy (the $\widetilde{O}$ hides logarithmic factors.)

Flajolet et al. in [9] proposes the $UCB - Simplex$ method (will be described in the next section), which has logarithmic regret. They analyze this algorithm for three different cases in terms of the number of resources: a single resource, two resources where one of them is time (the time consumption for each arm is deterministic and equal), and an arbitrary number of resources (in the third case, the costs for each arm are deterministic). The single resource case is most relevant to our problem, and it has regret $O(\ln B)$.

All of the aforementioned results are based on the assumption that the reward and cost(s) for each arm are stochastic and the environment remains unchanged. However, we believe that this assumption does not accurately model practical use cases. We are investigating this problem for different non-stochastic settings.

The first non-stochastic model is the classic adversarial model defined in [4]. In this model an adversary produces the reward and cost at each round. Auer et al. in [4] consider the bandit problem with a single reward for each arm at each time step. In their setting the only resource is time (time horizon T is the budget) and each arm deterministically consumes one unit of the resource at each time step. They use the *Hedge* algorithm which guarantees that the regret in the expert problem (full feedback) is at most $O(\sqrt{T \ln m})$ and designed the *EXP3* algorithm where the regret is $O(\sqrt{Tm \log m})$ where m is the number of arms. There is no result when the resource consumption of each arm is also chosen by an adversary. In the next section, we will investigate whether in this version of the problem it is possible to compete against the best single arm.

Slivkins and Upfal in [10] consider a changing environment in which the expected reward of each arm can change according to a Brownian motion. Their problem is a special case of the restless bandit problem in which the expected reward of each arm is the state of that arm. They define $\mu_i(t) \in [0, 1]$ (expected reward of arm i) as the state of the arm i at time t. Then, at time $t + 1$, $X_i(t)$ which is a sample from $\mathcal{N}(0, \sigma_i)$ will be added to the $\mu(t)$ and determines the value of $\mu(t+1)$ (the value should be in $[0, 1]$, therefore the interval has reflecting boundaries). Volatility of the arm i denoted by σ_i is known to the algorithm. They propose algorithms for the state informed (in which the feedback not only contains the reward but also the state of the arm) and state oblivious versions of this problem. However, their work does not consider the problem in the presence of cost.

Another line of work focuses on Thompson sampling which is a Bayesian algorithm. Agrawal and Goyal in [1] prove that the expected regret of Thompson sampling for the stochastic multi armed bandit problem (without cost) is logarithmic in time horizon. They assume Bayesian priors for the mean reward of each arm by using Beta distributions (their analysis is prior-free). A very natural extension of this work for the budgeted version of this problem is to have tow different priors for each arm (one for cost and one for reward). In this case, we can sample both of the reward and cost of each arm from the corresponding Beta distributions and play the arm with the highest ratio of the reward to cost. Xia et al. in [12] propose this extension and prove that the distribution dependent regret bound of this algorithm is $O(\ln B)$. Agarwal and Goyal in [2] generalized the Thompson sampling algorithm for the stochastic contextual bandit problem with linear payoffs.

4 Stochastic Budgeted MAB

In this section, we generalize the Thompson Sampling algorithm presented in [1] for the stochastic MAB problem to the budgeted version of the problem in the presence of cost. Then, we run some experiments and show that this algorithm works better than other existing non-Bayesian (UCB based) methods for this problem. Xia et al in [12] independently, proposed the same algorithm and showed similar results in their work.

Firstly, we explain the Thompson Sampling for the stochastic MAB introduced in [1]: There are n different arms and there is a time horizon T. There exists an unknown distribution for each arm and the reward of that arm in each time step is an iid sample of its corresponding distribution. In this algorithm, for each arm i there are two attributes: S_i and F_i (initialized to zero) and the prior for the mean reward of arm i is $Beta(S_i+1, F_i+1)$. At each time step the algorithm samples one value from the prior of each arm and play the arm i with the highest sampled value. Then it observes the reward r of that arm as feedback, and updates the prior of that arm as follows: with probability r the value of S_i will be increased by one and otherwise (with probability $1-r$) the value of F_i will be increased by one.

The natural, generalization of the above algorithm for the budgeted version of the problem with two unknown distributions for each arm is to keep two Beta distributions for each arm as priors: one for reward and one for cost. Therefore, each arm i has four attributes: Sc_i, Fc_i, Sr_i, and Fr_i. Then, at each time step we can take two samples for each arm i: One sample from $Beta(Sr_i+i, Fr_i+1)$ for the reward of arm i and one sample from $Beta(Sc_i+i, Fc_i+1)$ for the cost of arm i. Then we should play the arm with the highest ratio of the sampled reward to sampled cost. The update step is exactly the same for each distribution. In other words, if after playing arm i at the last time step the observed value for reward and cost are r and c then with probability r (c) we increase Sr_i (Sc_i) by one and otherwise we increase Fr_i (Fc_i) by one.

4.1 Experiments

Firstly, we describe the other non-Bayesian algorithms used in our simulation. Then, we provide the results.

4.1.1 UCB1

We do not use this algorithm in our simulation but this is the basic idea of the other algorithms and we briefly explain it here. This algorithm was designed in [3] for the stochastic case when each arm only produces a reward (no cost) at each time step. It is a simple index based approach: after initialization step (playing each arm once), at each round UCB1 chooses the arm i maximizing $\bar{x}_i + \sqrt{\frac{2\ln n}{n_i}}$ where $\bar{x}_i$ is the empirical average of arm i, n is the total number of times that the arms has been played up to this point, and n_i is the number of times that arm i has been played so far.

Note that, the first term $\bar{x}_i$ is the exploitation term encouraging playing arms with high estimated reward so far and the second term $\sqrt{\frac{2 \ln n}{n_i}}$ is the exploration term encouraging playing arms which has been played fewer so far.

The idea used in this algorithm is called optimism in the face of uncertainty. For each arm i, $\bar{x}_i + \sqrt{\frac{2 \ln n}{n_i}}$ is the highest value (optimistic estimate) of a high probability confidence interval for the expected reward of that arm. This value is obtained from the Hoeffding's inequality. Note that, changing the probability of the confidence interval results in other UCB algorithms (e.g. UCB2).

4.1.2 UCB-Simplex

This algorithm introduced by Flajolet et al. in [9]. The basic idea of this method consists of two main steps. In the first step, the algorithm finds an optimal basis of arms by solving an LP using the simplex method. In the second step, the algorithm must decide which of the arms in the optimal basis to play.

The LP is as follows:

$$\max \sum_{k=1}^{K} (\bar{r}_{(k,t)} + \beta \cdot \epsilon_{k,t}) \cdot \xi_k$$

$$\text{subject to } \sum_{k=1}^{K} \xi_k \cdot \bar{c}_{i(k,t)} \leq B(i) \text{ for each resource } i$$

$$\xi_k \geq 0 \text{ for each arm } k$$

Here, $\bar{r}_{(k,t)}$ and $\bar{c}_{i(k,t)}$ are the empirical estimates of the reward, and cost of resource i for arm k at time step t. The parameter β determines how much exploration there is and ξ_k denotes the number of times that arm k has been played so far. Lastly, $\epsilon_{k,t} = \sqrt{\frac{2 \ln(t)}{n_{k,t}}}$, which is obtained from the Hoeffding's inequality when the probability is $1 - \frac{2}{t^3}$ ($n_{k,t}$ is the number of times arm k has been played until time step t). Note that this is the same with the exploration term in UCB1.

Solving this LP is a general form technique. In our problem, since we are dealing with only one resource, there is only one restraint so there is no need to solve a LP. Similarly, for the second step there will only be one arm in the optimal basis. Since there is one resource, we will omit the index i notation.

Algorithm Description:

- Set $\beta = 1 + \frac{1}{\lambda}$, where λ is a lower bound on the average cost of any arm.
- Exploration phase: pull each arm in a round robin fashion until the empirical cost for each arm is non-zero.
- Exploitation phase: pull the arm with the largest UCB $= \frac{\bar{r}_{(k,t)} + \beta \cdot \epsilon_{k,t}}{\bar{c}_{(k,t)}}$.

4.1.3 UCB-BV2

This algorithm is introduced by Ding et al. in [7]. After initialization step (pulling each arm once) the index of arm i at time t is defined by the following formula:

$$D_{i,t} = \frac{\bar{r}_{i,t}}{\bar{c}_{i,t}} + \frac{1}{\lambda_t}(1 + \frac{1}{\lambda_t - \sqrt{\frac{\ln(t-1)}{n_{i,t}}}})\sqrt{\frac{\ln(t-1)}{n_{i,t}}}$$

where $\bar{r}_{i,t}$ and $\bar{c}_{i,t}$ are the empirical average reward and cost of arm i, $n_{i,t}$ is the number of times that arm i has been played so far, and λ_t is the minimum empirical cost among the arms at time t. They tried to make the algorithm independent of any knowledge about the distributions by using this value instead of the lower bound on the mean cost of the arms (like UCB-simplex) . At each step, algorithm pulls the arm with the highest index until it exhausts the budget.

4.1.4 BWK

In this algorithm we use the upper confidence bound for the reward and lower confidence bound for the cost of each arm and pulls the arm with the highest ratio. This is the natural extension of optimism in the face of uncertainty that we have seen in UCB1 to the budgeted version of the problem. We use the name BWK because the usage of lower and upper confidence bound is similar to [5]. However, that work consider a more general setting and their algorithm is more complicated.

4.1.5 Results

- Simulation 1: This simulation is similar to the simulations of [7].There are 10 different arms: The reward of each arm is drawn from a Bernoulli distribution, and the parameter p_i (probability of success) for each arm i is set randomly at the beginning of the simulation. The cost of each arm is drawn from a multinomial distribution and is from the set $\{0, 1/100, 2/100 \ldots 1\}$. The parameters of the multinomial distribution of each arm is also set randomly for each arm. We run this simulation 50 times and plot the average regret for each of the algorithms:

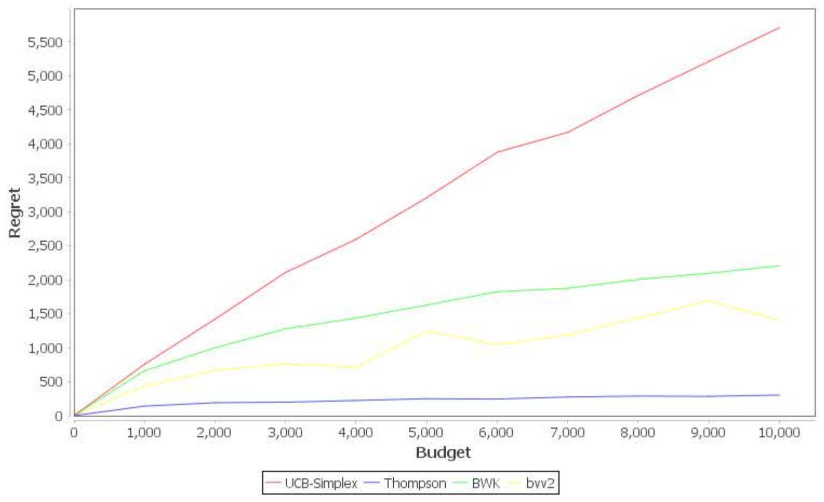


Figure 1: Simulation 1

- Simulation 2: In this simulation we consider 3 arms. For the first arm the reward at each step is 0.6 with probability 0.5 and 0.3 otherwise. For the second arm the reward at each time step is 0.8 with probability 0.7 and 0.1 otherwise. For these two arms the cost is always $1 - reward$. For the third arm the cost and reward are independent: the reward is 0.7 with probability 0.8 and 0.3 otherwise. The cost for this arm is 0.2 with probabilty 0.9 and 0.1 otherwise.

  We run the simulation for 50 times and the average regret of different algorithms for different budget values are as follows:

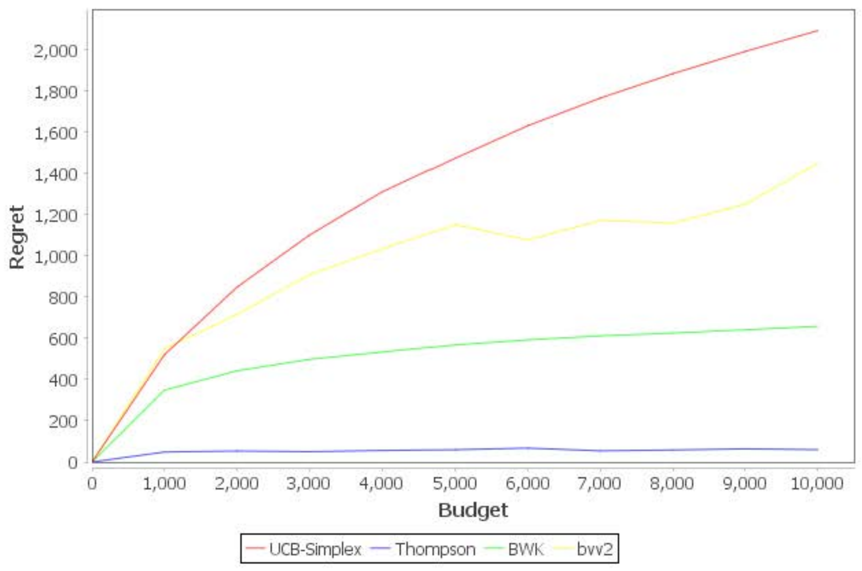


Figure 2: Simulation 2

As we can see in both of the plots Thompson Sampling algorithm has a lower regret than the other UCB based algorithms.

5 Adversarial Budgeted MAB

In this section, we study the adversarial model in the presence of cost. As we have mentioned earlier, the current algorithms for adversarial case only works with a time horizon which is a very special case of a cost. This section contains two parts. First, we consider the fully adversarial model which is the conventional adversarial model in which the reward and cost of each arm at each time step are determined by an advesary. Then, we introduce a new model which is a restricted adversarial model called leaky bucket input model.

5.1 Fully Adversarial Model

In this problem, there are m different arms and a single budget B, playing an arm produces a reward and incurs a cost. At each time step, we can choose one arm until sum of the cost of chosen arms over the time steps exceeds the budget. The reward and cost at each step can be any value (determined by an adversary) in $[0, 1]$. We assume that the feedback at the end of each time step reveals performances of all of the arms at the last time step (expert problem), and the adversary is oblivious. We show some negative results for this full feedback version which is easier than MAB problem.

Lemma 1. *No algorithm can obtain more than $\frac{1}{n}OPT$ where OPT is the optimal single arm policy.*

Proof. Consider the following simple example: in the first $\lfloor B \rfloor$ time steps for all of the arms the cost is 1 and reward is 0. Then in the time step $\lfloor B \rfloor + 1$ only the reward of one of the arms is 1 while its cost is 0. For the other arms the reward and cost are still 0 and 1, respectively. Even if the algorithm knows the structure of input, it cannot guarantee more than $\frac{1}{n}$ of the OPT and the reason is that it should choose one of the arms randomly. □

From the example in the proof of the previous lemma, it seems helpful to relax the budget constraint. In other words, the algorithm can spend a budget which is more than the actual budget (a function of the actual budget) and it should compete against the best single arm that consumes the actual budget.

If the algorithm is allowed to spend nB it is obvious that it can compete against the best single arm with budget B (we can run the weighted majority algorithm and ignore each expert after it consumes budget B in the previous rounds).

Now, we want to consider the case where a constant c is given and the relaxed budget is cB. The algorithm should compete against the best single arm with budget B. In the next lemma we show that it is not possible to have an algorithm that spends cB budget and guarantees a reward which is a constant ratio of the reward of the best single arm.

Lemma 2. *Constant values c_1 and c_2 are given. Let OPT be the reward of the best single arm with budget B. It is not possible to design a policy which guarantees $\frac{1}{c_1}OPT$ reward with c_2B budget.*

Proof. We prove it by contradiction: assume that c_1 is integer (otherwise consider $\lceil c_1 \rceil$). If there exists such an algorithm, it must work on every input. Consider the following n different inputs: the first expert only has non-zero reward and cost for the first $2B$ steps such that in B of them the reward is 1 and the cost is 0 and in the other B steps the reward is 0 and the cost is 1. In the ith input the last $n - i$ experts always have cost 0 and reward 0. For every $2 \leq j \leq i$ the jth expert produces non-zero reward and cost only at the time steps between $S_j = (2c_1+1)^{j-2}B+(j-1)B+1$ and $E_j = (2c_1 + 1)^{j-1}B + jB$ such that in B of them it has reward 0 and cost 1 and in the other $(2c_1 + 1)^{j-2} \times 2c_1B$ steps it has reward 1 and cost 0.
The algorithm should results in a desirable outcome for all of the n different inputs. We can see that it should choose expert i in at least $\frac{1}{2c_1}$ fraction of the times that this expert has non-zero reward or cost. The reason is that the algorithm should work for the ith input, and for each $j > i$, the ith input and jth input are the same before time step E_i. Therefore, for the nth input the expected cost will be $\frac{n}{2c_1}B$ which is not a constant factor of the budget. □

Note that in our proof the adversary only chooses one of the two possibilities for each arm at each time step: reward 1 with cost 0 and vice versa. It is worthwhile to mention that the structure of the proof is similar to [8].

5.2 Leaky Bucket Input Model

As we showed earlier, fully adversarial model is too strong. In this part, we introduce a new input model called Leaky bucket input model which is a weaker version of the adversarial model.

Input Model:

- Each arm i has four unknown values: (r_i, a_i) and (c_i, b_i).
- Assume $r_i, c_i \in [0, 1]$ for each arm i.

- For each arm i, until time t (any window of time starting from time 0), the total reward $\in [r_i t - a_i, r_i t]$, and total cost $\in [c_i t, c_i t + b_i]$.

Here, it is possible for a_i and b_i to be functions of time. However, we consider the simple case where these are fixed. Note that this case is similar to stochastic version in which the expected value of the reward and cost of arm i are r_i and c_i, respectively. However, at each step playing an arm is not an independent sample from a joint distribution with those expected values. An adversary can choose the values but we restrict the power of the adversary by parameters a_i and b_i. In other words, the adversary cannot deviate much from the actual values of the arms.

First, note that sampling the arms like the other input models is not helpful anymore. The reason is that the adversary can control the input. For example, the adversary can choose a number of small cost followed by a very large cost for an arm. In this case we cannot use Hoeffding's inequality and other similar techniques. The only option is to consecutively play an arm for some time steps in order to find the expected values. Since we do not know the parameters for each arm, the best option is to play each of them until it consumes a certain amount of budget. After this sampling step we will play the best arm according to sampling step for the rest of the time steps. We propose a simple policy for this model:

Like before, let B be the total budget, and let m be the total number of arms. Let ϵ be a parameter.

Algorithm Description:

- Exploration: Play an arm continuously until $\frac{\epsilon B}{m}$ of cost is consumed for that arm. Then switch to another arm and do the same until all arms are explored.

- Exploitation: Pick the arm with the highest $\frac{\bar{r}}{\bar{c}}$, and play this arm for the remainder of budget.

The value of ϵ will be set so as to minimize the regret. A similar exploration and then exploitation policy has been defined in [11] for the stochastic version of the problem. One possible future direction is to study the problem when a_i and b_i are functions of time (instead of constant values).

6 Recency Policy: Detecting the Changes in the Environment

As mentioned earlier, we are interested in MAB problems in a non-stochastic setting. We considered adversarial settings in the previous section. In this section, we consider another non-stochastic setting called Markovian input model. In this setting, each arm might have multiple states, and each state has an associated reward and cost. There are transition probabilities specifying the transition of arms from a state to another.

We propose Recency which is a new algorithm to solve the problem: as the name may suggest, this algorithm gives more importance to recent events in the cost and reward sequence. The intuition behind this is as follows: in certain settings, the characteristics of the arms may change over time, thus it may be inefficient and inaccurate to consider older information. The algorithm has some sample and update steps. In any time step which is not a sample step or update step, the algorithm chooses the arm picked in the last update step. In sample steps, it randomly samples the rewards and costs for different arms and updates their empirical estimates. In each update step, it uses those estimated values to pick the arm with the highest ratio of reward to cost. In addition, it reset (or multiply by a discount factor) the previous estimated values for the arms. The sampling and update steps depend on the budget consumed, but at random points based on a sampling from an exponential distribution. We define the following two quantities: sampling rate $= s = \frac{B^{\frac{-1}{3}}}{m}$, and update rate $= u = B^{\frac{-2}{3}}$.

Algorithm Description:

- Sample Time (determined by sampling rate s)
  - Randomly choose arm k
  - Update the total reward r_k, total cost c_k, and number of plays n_k.
  - Set the next sample time to when the cost consumed is *Exponential(s)*.
  - (The number of sampling steps is roughly $B^{\frac{2}{3}}/m$).
- Update Time (determined by update rate u)
  - Choose arm i with the largest $\frac{r_i}{c_i}$. Use this arm until the next update.
  - For all arms k, reset r_k,c_k, and n_k to 0.
  - Set the next update time to when the cost consumed is *Exponential(u)*.
  - (The number of updating steps is roughly $B^{\frac{1}{3}}$).

As opposed to algorithms designed for a stochastic setting, this algorithm can detect changes in the states of the arms. Additionally, since the sampling frequency depends on the budget consumed, it will not spend too much budget on sampling. However, since it periodically sample all the arms it is obvious that its regret can be larger than the algorithms that do not consider changing environment in some cases.

6.1 Experiments

Here, we consider different scenarios and compare the regret of our algorithm with UCB-Simplex (described in stochastic MAB section).

Below you can see the results of the simulations comparing UCB-Simplex and Recency:

For each state, the first entry is the reward, and the second entry is the cost. The transition probability shows the probability of a change from one state to the other one. In the plots, the x-axis are the different budgets, and the y-axis represents the total regret with regards to OPT.

- Simulation A:

  In this simulation we have 2 arms.

Arm	State 1	State 2	Transition Probability
1	(0.5, 0.5)	-	
2	(0.1, 1)	(1, 1)	S1 $\to$ S2: 0.0001, S2 $\to$ S1: 0.001

- Simulation B:

  In this simulation we have 3 arms. The first two arms are the same with the two arms of simulation A. The third one with small cost and reward is added to the options:

Arm	State 1	State 2	Transition Probability
1	(0.5, 0.5)	-	
2	(0.1, 1)	(1, 1)	S1 → S2: 0.0001, S2 → S1: 0.001
3	(0.005, 0.05)	(0.05, 0.05)	S1 → S2: 0.0001, S2 → S1: 0.001

Figure 1 shows the results of the simulations A and B :

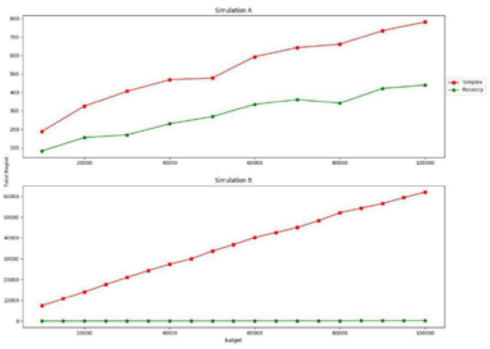


Figure 3: Simulations A and B.

Although the two scenario are almost the same (the extra arm has the same ratio of reward to cost with the second one), regret of UCB-Simplex is very different for these scenarios. The reason is that the known lower bound is much smaller for the second scenario and even for large value of budget the algorithm explore in most of the time steps. Recency periodically sample the arms but it does not depend on the lower bound. In this scenario, the Recency behave the same for the two scenarios.

- Simulation C:

  In this simulation we have 2 arms:

Arm	State 1	State 2	Transition Probability
1	(1, 1)	(0.3,0.6)	0.0001
2	(0.9, 1)	(1, 0.7)	0.0001

- Simulation D:

  In this simulation we have 2 arms which are the same and start from different states:

Arm	State 1	State 2	Transition Probability
1	(1, 0.1)	(0.1,1)	0.001
2	(0.1, 1)	(1, 0.1)	0.001

Figure 2 shows the results of the simulations C and D.

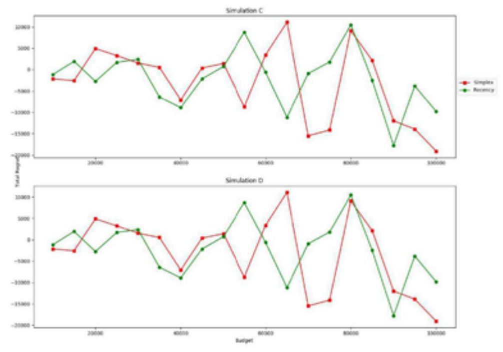


Figure 4: Simulations C and D

- Simulation E:

  In this simulation we have four arms:

Arm	State 1	State 2	Transition Probability
1	(1.0, 0.01)	(0.01, 1.0)	0.001
2	(0.9, 0.1)	(0.1, 0.9)	0.001
3	(0.8, 0.15)	(0.15, 0.8)	0.001
4	(0.7, 0.2)	(0.2, 0.7)	0.001

- Simulation F:

  In this simulation there are two arms:

Arm	State 1	State 2	Transition Probability
1	(1, 0.0001)	(0.1, 1)	0.0001
2	(0.01, 1.0)	(1, 1)	0.0001

Figure 3 shows the results of the simulations E and F.

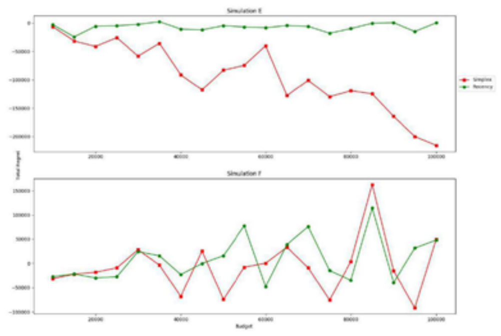


Figure 5: Simulations E and F

As we can see in scenarios C,D,F the performance of the two algorithms are similar. In scenario E, UCB-Simplex has a lower regret. As we mentioned earlier since our algorithm sample all the arms periodically, it is not surprising that in some case UCB-Simplex has a lower regret. However, it seems that our algorithm can detect the changes in some cases. In addition, it does not depend on the lower bound of the cost consumption which can be problematic in some cases like scenario B.

7 Contextual MAB with Linear Payoffs

In this section, we generalize the existing Thompson sampling algorithm for a contextual bandit problem with linear payoffs, explained in [2], to another contextual bandit problem with linear payoffs. Firstly, we explain the first problem and algorithm presented in that work. Then, we explain the other problem and our idea for using the Thompson sampling algorithm for it. Finally, we implement the suggested algorithm for the second problem and show that it has a low regret by some experiments.

In [2], at each time step t, a context $b_i(t) \in \mathbb{R}^d$ is given for each arm i (by an adversary). In addition, there is an unknown parameter $\bar{\mu} \in \mathbb{R}^d$ for the problem. The reward of playing arm i at time t is $b_i(t)^T\bar{\mu}$ plus a noise which is a sample from a R-sub-Gaussian distribution. The goal is to compete with the predictor who knows the parameter $\bar{\mu}$ and plays the arm i maximizing $b_i(t)^T\bar{\mu}$ at time step t. In the Thompson sampling algorithm proposed for this problem, at each time step they sample $\tilde{\mu}(t)$ from a multivariate Gaussian distribution $\mathcal{N}(\hat{\mu}, v^2B^{-1})$ and play the arm with the highest $b_i(t)^T\tilde{\mu}(t)$. The initial values are as follows: $B = I_d$, $\hat{\mu} = f = 0_d$. After playing such an arm i and observing reward r_t, the update step is as follows: $B = B + b_i(t)b_i(t)^T$, $f = f + b_i(t)r_t$, and $\hat{\mu} = B^{-1}f$.

We wish to generalize the mentioned Thompson sampling algorithm to the following problem: At each time step, a single context $b(t) \in \mathbb{R}^d$ is given which is drawn from any distribution (it is not chosen by an adversary anymore). Each arm i has an unknown parameter $\bar{\mu}_i \in \mathbb{R}^d$. The reward of playing arm i at time t is $b(t)^T\bar{\mu}_i$ plus a noise which is a sample from a R-sub-Gaussian distribution. Note that in this problem the context specifies features of the environment at that time step (not features of the arm). This is the reason that we have only one context vector at each time step. This problem was defined in [6]. They provide an interesting motivation for this problem. In their work they mention that each arm can be a special treatment and the context might represent the characteristics of the patient at that time step. This example justifies the assumption that there is a single context at each time step which is drawn from a distribution. This problem is harder than the previous one because for each arm there is an unknown parameter and we need to estimate all of them and at each time step we can only update one of the estimated values.

In order, to generalize the proposed algorithm for the new setting we have one multi-variated Gaussian prior for each of the arms and take a sample, $\tilde{\mu}_i(t)$, for each arm i at each time step t. Then we play the arm i which maximizes $b(t)^T\tilde{\mu}_i(t)$ and observe the reward r. Finally, we update the prior of the arm i just as before. We implement this algorithm and the results are shown in the experiments subsection.

7.1 Experiments

Firstly, we describe the intuition behind OLS algorithm which is presented in [6]. They prove that the expected regret of this algorithm is in $O(d^2 \log^{\frac{3}{3}} d. \log T)$.

7.1.1 OLS

In this algorithm, they use OLS estimator to find the estimated parameter for each arm i. Let $X_i \in R^{n*d}$ be the contexts in samples that arm i has been played and $Y_i \in R^n$ be the rewards of those samples. Then the estimated parameter $\beta^i_{X_iY_i}$ is defined as follows:

$$\beta_{iX_iY_i} = (X_i^T X_i)^{-1} X_i^T Y_i$$

They use two kind of samples for each arm: forced samples and all samples. The forced sample of each arm are determined by the algorithm upfront (before playing any arms) and all sample contains all the time steps that arm has been played so far. The algorithm uses forced samples of each arm to choose a subset of arms which has "high enough" estimated reward at this time (the reward is obtained by using the OLS estimator of the forced samples of each arm and the context of this step). Then, it uses all samples of each arm and play the arm in that subset with the highest estimated reward at this time step.

7.1.2 Results

7.1.3 Results

- Simulation 1: The dimension of the context vector at each time step is 6 and each of the entries at each time step is generated uniformly at random (the entries are independent of each other).

  There are 6 different arms at this simulation and the parameter of each arm is generated uniformly at random at the beginning of the simulation. We run the simulation for 10 times and the following plot shows the average regret of the different methods for different time horizons.

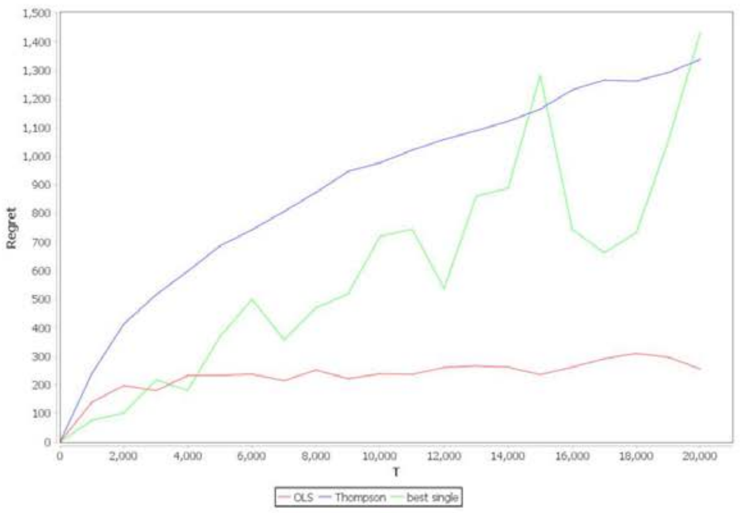


Figure 6: Simulation 1

- Simulation 2: The dimension of the context vector at each time step is 6 and each of the entries at each time step is generated uniformly at random (the entries are independent of each other).

  There are 6 different arms at this simulation and the parameter of arms are as follows: $\mu_0 = \{1, 1, -1, -1, 0, 0\}; \mu_1 = \{1, -1, 1, -1, 0, 0\}; \mu_2 = \{-1, -1, -1, 0, 1, 1\}; \mu_3 = \{1, -1, 0, 0, -1, 1\}; \mu_4 = \{0, 0, -1, -1, 1, 1\}; \mu_5 = \{0.1, 0.1, 0.1, 0.1, 0.1, 0.1\}$

  We run the simulation for 10 times and the following plot shows the average regret of the different methods for different time horizons.

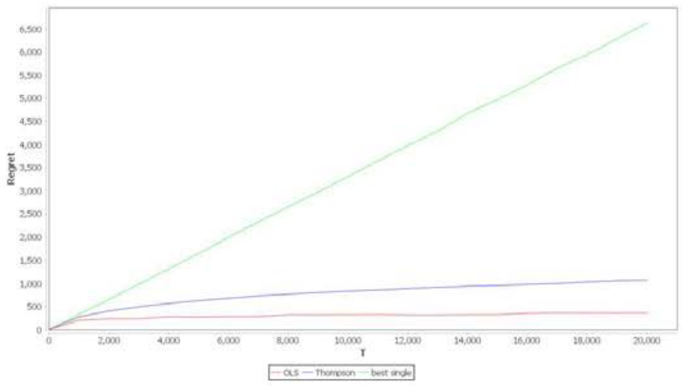


Figure 7: Simulation 2

- Simulation 3: The dimension of the context vector at each time step is 6. Each entry is generated independently from a normal distribution at each time step. The average of the normal distribution of each dimension is generated uniformly at random at the beginning. The variance of the distribution is also generated uniformaly at random (between 0 and 0.5).

  There are 6 different arms at this simulation and the parameter of arms are as follows: $\mu_0 = \{1, 1, -1, -1, 0, 0\}$; $\mu_1 = \{1, -1, 1, -1, 0, 0\}$; $\mu_2 = \{-1, -1, -1, 0, 1, 1\}$; $\mu_3 = \{1, -1, 0, 0, -1, 1\}$; $\mu_4 = \{0, 0, -1, -1, 1, 1\}$; $\mu_5 = \{0.1, 0.1, 0.1, 0.1, 0.1, 0.1\}$

  We run the simulation for 10 times and the following plot shows the average regret of the different methods for different time horizons.

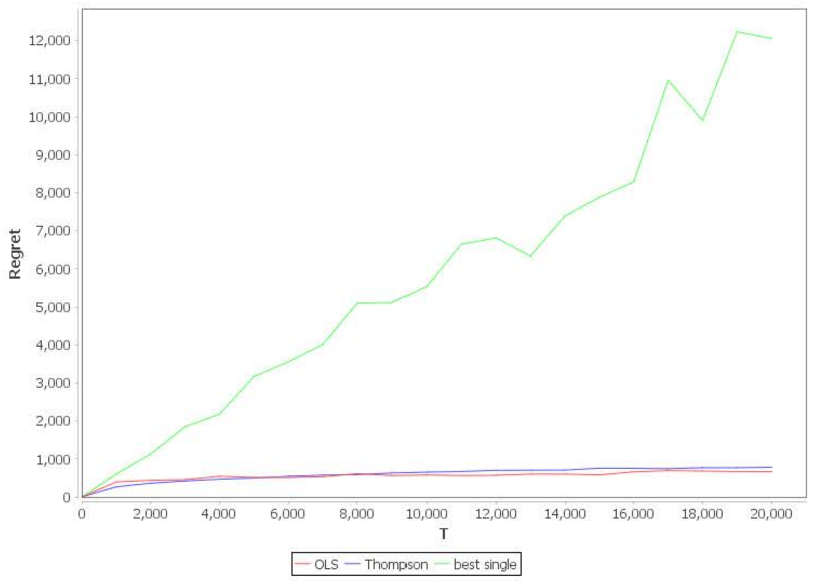


Figure 8: Simulation 3

- Simulation 4: The dimension of the context vector at each time step is 6. Each entry is generated independently from a normal distribution at each time step. The average of the normal distribution of each dimension is generated uniformly at random at the beginning. The variance of the distribution is also generated uniformaly at random (between 0 and 0.5).

  There are 6 different arms at this simulation and the parameter of each arm is generated uniformly at random at the beginning of the simulation. We run the simulation for 10 times and the following plot shows the average regret of the different methods for different time horizons.

  We run the simulation for 10 times and the following plot shows the average regret of the different methods for different time horizons.

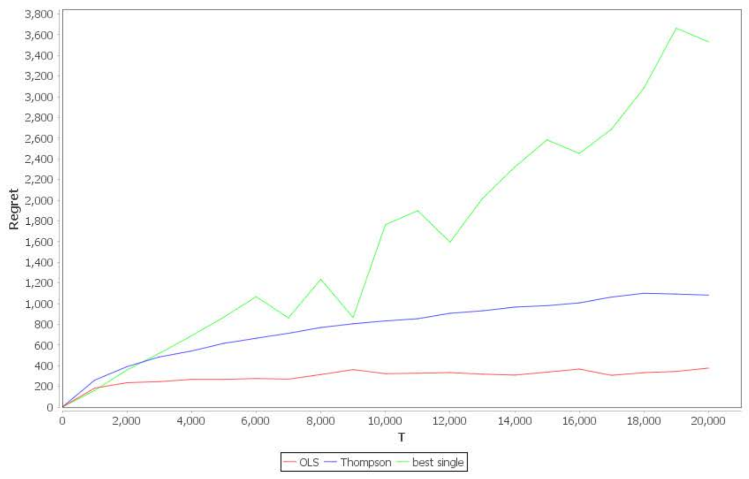


Figure 9: Simulation 4

8 Open Problems

We have introduced the leaky bucket input model. The definition is as follows: for each time t and each arm i with parameters (r_i, a_i) and (c_i, b_i) the total reward and cost of that arm in the first t time steps should be in $[r_i t - a_i, r_i t]$ and $[c_i t, c_i t + b]$, respectively. We proposed a simple policy for the case that a_i and b_i are constant values. Extending this policy to the case that a_i and b_i are functions of time is an open question. In this case, it seems that for the arms that we sample later we need more samples. It is obvious that the problem cannot be solved for any function because it includes the fully adversarial setting. Specifying the functions that a policy can compete against the single best arm for this model is a possible future line of work.

Another important future line of work is the Markovian input model. We provide the Recency algorithm and compare it with the UCB-Simplex by some experiments. Providing a theoretical analysis for this algorithm or for another algorithm showing that the regret guaranteed to be low is an important future direction for this problem. Also comparing the two algorithms with real data of a changing environment might be helpful to decide which of them works better in real situations.

References

[1] S. Agrawal and N. Goyal. Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on Learning Theory*, pages 39–1, 2012.

[2] S. Agrawal and N. Goyal. Thompson sampling for contextual bandits with linear payoffs. In *International Conference on Machine Learning*, pages 127–135, 2013.

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[4] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 322–331. IEEE, 1995.

[5] A. Badanidiyuru, R. Kleinberg, and A. Slivkins. Bandits with knapsacks. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 207–216. IEEE, 2013.

[6] H. Bastani and M. Bayati. Online decision-making with high-dimensional covariates. 2015.

[7] W. Ding, T. Qin, X.-D. Zhang, and T.-Y. Liu. Multi-armed bandit with budget constraint and variable costs. In *AAAI*, 2013.

[8] E. Even-Dar, M. Kearns, and J. Wortman. Risk-sensitive online learning. In *International Conference on Algorithmic Learning Theory*, pages 199–213. Springer, 2006.

[9] A. Flajolet and P. Jaillet. Logarithmic regret bounds for bandits with knapsacks. *arXiv preprint arXiv:1510.01800*, 2015.

[10] A. Slivkins and E. Upfal. Adapting to a changing environment: the brownian restless bandits. In *COLT*, pages 343–354, 2008.

[11] L. Tran-Thanh, A. Chapman, J. E. Munoz De Cote Flores Luna, A. Rogers, and N. R. Jennings. Epsilon–first policies for budget–limited multi-armed bandits. 2010.

[12] Y. Xia, H. Li, T. Qin, N. Yu, and T.-Y. Liu. Thompson sampling for budgeted multi-armed bandits. In *IJCAI*, pages 3960–3966, 2015.

APPENDIX C Sample efficient PAC Reinforcement Learning

Sample Efficient PAC Reinforcement Learning in MDPs via Inter and Intra-task Knowledge Transfer

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

While PAC algorithms for reinforcement learning provide strong theoretical guarantees, their high sample complexity has thus far prevented their use in practical applications. Transfer learning has the potential to reduce the sample complexity of learning by reusing samples across the state-action-MDP space, yet few PAC algorithms exist that take advantage of transfer learning. We introduce a unified approach to inter- and intra-task knowledge transfer with PAC guarantees which, under appropriate conditions, leads to a sample complexity reduction that is exponential with respect to the dimensionality of the state-action-MDP space. We show that approximate local linearity is sufficient (but not necessary) for our algorithm to yield dramatic sample complexity reductions over the current state of the art in PAC reinforcement learning. In addition we show that, in the batch-mode learning setting, pessimism in the face of uncertainty can offer significant advantages over optimistic or maximum-likelihood estimates.

1 Introduction

Reinforcement learning (RL) is suffering from a disconnect between theory and practice. Algorithms with strong theoretical guarantees do not perform well in practice [4], while the algorithms that perform best in practice offer no theoretical guarantees (in fact counter-examples for many of these algorithms exist, which show that they can have very poor worst-case performance [13]).

Probably approximately correct (PAC) algorithms for RL are a class of algorithms that has recently garnered significant attention. They offer a high probability guarantee (probably), that they will produce a policy that performs almost as well (approximately) as the optimal policy (correct), while requiring a number of samples that is at most polynomial to the parameters of the problem. The polynomial guarantee is with respect to 1) the size of the state-action-MDP space, 2) the planning horizon, 3) how far can the approximate policy deviate from the optimal policy, and 4) the probability that the algorithm will fail to produce an acceptable policy given a sufficient number of samples.

Despite the significant step forward that the polynomial sample guarantees of PAC-RL algorithms represent when compared to the guarantees provided by their predecessors (asymptotic guarantees where convergence is only guaranteed in the limit of infinite samples), their sample requirements have proven to be too steep for practical applications. Lower bound results [22, 10, 1] indicate that providing PAC guarantees is inherently sample-intensive in the worst case. Recent research has focused on identifying classes of "well-behaved" problems, for which far fewer samples suffice to provide PAC guarantees [19].

The size of the state-action-MDP space is by far the largest contributor to the sample complexity of RL. As the number of state, action, and MDP variables increases, the size of the space can grow exponentially with the number of dimensions. The problem is even more pronounced in the case of

Submitted to 31st Conference on Neural Information Processing Systems (NIPS 2017). Do not distribute.

continuous spaces. Inter and intra-task knowledge transfer aims to reduce the effective size of the state-action-MDP space by reusing samples across state-actions in separate MDPs or within the same MDP respectively.

To the best of our knowledge, only one algorithm exists that can take advantage of intra-task knowledge transfer and maintain PAC guarantees under appropriate conditions [2]. Similarly, only one algorithm is known to exist that addresses inter-task transfer with PAC-guarantees [18]. We present a general transfer learning algorithm for which the two previous known results are special cases. This new algorithm can take advantage of both inter and intra-task knowledge transfer without having to differentiate between the two, and is able to maintain PAC guarantees under more general conditions than its predecessors. An approximate local linearity assumption on the dynamics of the process, is a sufficient (but not necessary) condition for our algorithm to yield sample complexity reductions that are exponential with respect to the dimensionality of the state-action-MDP space.

For simplicity of exposition we will show how our approach to transfer learning can be used to reduce sample complexity in the batch mode learning setting (algorithms in this setting learn from a fixed set of samples and perform no learning during the policy execution or testing phase). If we combine our approach with recent PAC exploration algorithms [18, 14], the sample complexity reduction in the exploration setting is identical to the improvement in the batch mode learning setting. Additionally, we will assume that the task variation is observable. Our algorithm can be used with minor modifications as a subroutine to improve the sample complexity of recent task identification algorithms [3, 14].

Optimism in the face of uncertainty has become the norm for exploration algorithms, and maximum likelihood estimation has long been the norm for batch-mode learning algorithms. By contrast, our algorithm employs pessimism in the face of uncertainty. We show that in the context of batch-mode learning, pessimism in the face of uncertainty can offer significant advantages over optimistic or maximum-likelihood estimates.

2 Background, notation and definitions

In the following, important symbols and terms will appear in **bold** when first introduced. Let $\mathcal{X}$ be the domain of x. Throughout this paper, $\forall x$ will serve as a shorthand for $\forall x \in \mathcal{X}$. We will use $\boldsymbol{x}, \boldsymbol{\bar{x}}, \boldsymbol{\tilde{x}}, \boldsymbol{x'}$ to denote time-indexed state-action-MDP triples, where $\boldsymbol{x_h, x_s, x_a, x_m}$ denote the time-step, state, action, and MDP components of x.

A *Markov decision process* (MDP) family is a 7-tuple $(\mathcal{S}, \mathcal{A}, M, P, R, \gamma, H)$, where $\boldsymbol{\mathcal{S}}$ is the state space of the process, $\boldsymbol{\mathcal{A}}$ is the action space, $\boldsymbol{M}$ is a family of MDPs, $\boldsymbol{P}$ is a Markovian transition model $\big(p(x'_s|x)$ denotes the probability density of a transition to state x'_s when starting from state-action-MDP triple x at time $x_h\big)$, $\boldsymbol{R}$ is a reward function $\big(R(x, x'_s)$ is the reward for transitioning to state x'_s from $x\big)$, $\boldsymbol{\gamma} \in [0, 1]$ is a discount factor for future rewards, and $\boldsymbol{H}$ is a horizon time for each episode. In this paper we will be considering time dependent transition and reward models. A *deterministic policy* $\boldsymbol{\pi}$ for an MDP is a mapping $\pi : \mathcal{S}, h \mapsto \mathcal{A}$ from states and time-steps to actions; $\pi(x_s, h)$ denotes the action choice in state x_s at step h. The value $\boldsymbol{Q^{\pi}(x)}$ of a state-action-MDP triple x at step x_h under policy π is defined as the expected, accumulated, discounted reward from x at step h, when all decisions starting from step $h+1$ are made according to policy π. There exists an optimal policy $\boldsymbol{\pi^*}$ for choosing actions which yields the optimal value function $\boldsymbol{Q^*(x)} = \max_\pi Q^\pi(x)$. For a fixed policy π the Bellman operator for Q is defined as: $\boldsymbol{B^{\pi}Q(x)} = \int_{x'} p(x'|x,\pi)\Big(R(x, x'_s) + \gamma Q(x')\Big)$. The value of a state s at step h under policy π in MDP m is defined as the expected, accumulated, discounted reward from s in MDP m at step h, when all decisions starting from step h are made according to policy π, $\boldsymbol{V^{\pi}(s, m, h)} = \int_x P(x|x_s = s, x_m = m, x_h = h, \pi^{\bar{Q}})Q^{\pi^{\bar{Q}}}(x)$. We also define $\boldsymbol{V^*(s, m, h)} = \max_\pi V^\pi(s, m, h)$.

In reinforcement learning [23], a learner interacts with a stochastic process modeled as an MDP and observes the state at every step, however the transition model P is not known. The goal is to learn a near optimal policy using experience collected through interaction with the process. At each step of interaction, the learner observes the current state x_s and MDP x_m, chooses an action x_a, and observes the resulting next state x'_s, essentially sampling the transition model of the process. Thus experience comes in the form of (x, x'_s) samples.

For simplicity of exposition we will assume that the reward function is known. Our results easily generalize to the unknown reward setting. We will also assume that all rewards are bounded, and without loss of generality that they lie in $[\mathbf{0}, \boldsymbol{R}_{\mathbf{max}}]$.[1] We will use $Q_{\max}$ to denote $\max_{x,h} Q^*(x)$ (the maximum expected discounted reward for any policy from any state-action-MDP triple at any timestep), and define $\hat{Q}_{\max} = R_{\max} + \gamma Q_{\max}$.

Definition 2.1. *The* **sample set $\tilde{\boldsymbol{S}}$** *is defined to be the set of all samples retained by our algorithm.*

Definition 2.2. $\boldsymbol{d(x, \bar{x})}$ *is defined to be the distance of (x_s, x_a, x_m, x_h) to $(\bar{x}_s, \bar{x}_a, \bar{x}_m, \bar{x}_h)$ in some well-defined distance metric. We also define the shorthand* $\mathbf{d}(\boldsymbol{x}, \bar{\boldsymbol{x}}, \boldsymbol{c}) = \max\{0, d(x, \bar{x}) - c\}$ *where c is a constant,* $\boldsymbol{d_k(x, \tilde{S})}$ *the distance of the k-th nearest sample in $\tilde{S}$ to x, and* $\boldsymbol{d_k(x, \tilde{S}, c)} = \max\{0, d_k(x, \tilde{S}) - c\}$. *When an insufficient number of samples exists to compute the last two functions, they return infinity.*

Examples of distance metrics include weighted norms, norms on linear and non linear transformations, as well as norms on features of the domain space. In the following we will use $\tilde{\boldsymbol{Q}}$ to denote the approximate value function computed by our algorithm, and $\pi^{\tilde{Q}}$ the greedy policy over $\tilde{Q}$. followed by our algorithm.

3 A motivating example

We will use an autonomous race-car agent as an example of how samples (knowledge) can be shared across different parts of the space, both within the same task (intra-task transfer), and among different tasks (inter-task transfer). The agent can be presented with multiple variations of a race car problem: 1) Different cars (four wheel drive, front wheel drive, rear wheel drive), 2) different tracks (gravel, dirt, clay, asphalt), and 3) different driving conditions (sun, rain, snow).

Consider the effects of applying the brakes for 0.3 seconds when traveling at 100 km/h. Suppose that the speed of the vehicle after applying the brakes is 80 km/h. It is reasonable to assume that if we apply the brakes at similar speeds, say 99.5 and 100.5 hm/h, the effects will be similar. How different the starting state can be before the similarity assumption starts to break down strongly depends on how we define the effects of an action. In the simplest case, the effect of an action at a particular state is the state that results from taking that action. In other words, we are saying that the car's speed after applying the brakes when starting at 99.5, 100, and 100.5 km/h should be about 80 km/h. One obvious way to make this assumption true over a larger range of values is to think of an action as a transformation. Instead of saying that the effect of applying the brakes at 100 km/h is a resulting speed of 80 km/h, we can say that the effect of applying the brakes at 100 km/h is a reduction in speed by 20 km/h. The range of values for which this transformation will be sufficiently accurate will depend on how closely this system resembles a linear system. For a perfectly linear system the transformation will be accurate over the entire range of values. Note that transformations are not required to be linear functions. For example, one could define a saturating transformation: Applying the brakes when starting from positive speed cannot result in negative speed.

Previous work on inter-task transfer has focused on transferring value functions [18], policies [24], or complete sample sets [3, 14]. While all the above strategies can be effective in certain situations, there exist very simple examples where they fail. Consider the case where the autonomous car agent has to complete races in the same track on three different days. On the first the weather is mild. On the second the track is covered with ice. On the third, the entire track is clear of ice except for a particular turn. While, up to the point of the ice-covered turn, the dynamics of the third task are identical to the dynamics of the first, entering the turn with the same speed that the agent was entering it on the first might cause the car to lose control. Both the value function and policy of the first task are not transferable to the third, even on segments of the track where these two tasks are identical. The value function and policy of the second task are similarly problematic since, even in the best case, they are too conservative for the third task, and, in the worst case, they are using drifting techniques that will not work on clear segments of the track. While the problem is trivially solvable by transferring samples from the first task for clear segments and samples from the second task for the icy segment, even PAC transfer learning methods that directly transfer samples do not have this flexibility. Existing methods only allow using all samples gathered from a particular MDP (or cluster of MDPs) or none at all (a notable exception in the non-PAC setting is the work of Lazaric et. al. [11]).

[1]It is easy to satisfy this assumption in all MDPs with bounded rewards by simply shifting the reward space.

4 Inter and intra-task knowledge transfer

Both inter and intra-task transfer can be performed via a transfer function:[2]

Definition 4.1. *A* **transfer function** $\boldsymbol{f_{tr}(x, x'_s, \bar{x}) \to \bar{x}'_s}$ *is a function that takes as input a source state-action-MDP triple, an associated next state, and a target state-action-MDP triple, and outputs a predicted next state for the target state-action-MDP triple.*

Two straightforward examples of transfer functions are 1) the identity transfer function $f_{tr}(x, x'_s, \bar{x}) \to x'_s$, and 2) the relative transfer function $f_{tr}(x, x'_s, \bar{x}) \to x'_s + \bar{x}_s - x_s$. Significantly more complex, non-linear, domain specific transfer functions can be defined, including transfer functions that translate functions between MDPs with different state and action spaces. We are using the identity and relative transfer functions as examples because they are simple and intuitive, and as we will see in the related work section, much of existing work in inter and intra-task transfer learning has implicitly or explicitly used these two transfer functions.

4.1 The transfer learning algorithm

Definition 4.2. *The* **cover set** $\boldsymbol{\tilde{C}(\tilde{S}, m)}$ *is defined to be the (possibly infinite) set of $x : x_m = m$ for which $\min_{\bar{x} \in \tilde{S}}\{d(x, \bar{x})\} < Q_{\max}$.*

The cover set is the portion of the state-action space for which at least one sample exists less than $Q_{\max}$ away.

Definition 4.3. *The* **discretization set** $\boldsymbol{\tilde{D}(\tilde{S}, m, d_Q)}$ *is defined as a discretization of the cover set $\tilde{C}(\tilde{S}, m)$, such that for every $x \in \tilde{C}(\tilde{S}, m)$ there exists $\bar{x} \in \tilde{D}(\tilde{S}, m, d_Q)$ such that $d(x, \bar{x}) \leq d_Q$. We also define the shorthand* $\boldsymbol{d(x, \tilde{D}(\tilde{S}, m, d_Q))} = \min_{\bar{x} \in \tilde{D}(\tilde{S}, m, d_Q)}\{d(x, \bar{x})\}$.

We will use $|\tilde{D}(\tilde{S}, m, d_Q)|$ to denote the number of elements in $\tilde{D}(\tilde{S}, m, d_Q)$ for which at least one sample in $\tilde{S}$ exists less than $Q_{\max}$ away. As we will see in our analysis, our algorithm's computational complexity depends on $|\tilde{D}(\tilde{S}, m, d_Q)|$ (which is finite), even if the cardinality of the state-action space is infinite. The best way to construct $\tilde{D}(\tilde{S}, m, d_Q)$ depends on the application. The simplest implementation is to find the extreme values for every state and action variable in $\tilde{C}(\tilde{S}, m)$ (this can be easily performed given $\tilde{S}$ and m) and allocate a dense multidimensional array covering all values within that space with appropriate step sizes to make sure that the radius of each element is less than or equal to d_Q. The advantage of this structure is that it offers $O(1)$ lookups. The disadvantage is that if the volume covered by $\tilde{C}(\tilde{S}, m)$ is significantly smaller than the volume contained within the extreme values, the dense scheme could lead to inefficient memory usage. A sparse representation (see for example the representation scheme employed by Pazis and Parr [18]) is space efficient, but can have significantly worse lookup performance ($O(|\tilde{D}(\tilde{S}, m, d_Q)|)$ for a naive implementation).

Given a set of samples and target MDP m, algorithm 1 generates a discretization set (line 2) and uses it to compute a pessimistic approximate value function (lines 3-7). It then follows the greedy policy over the computed value function (lines 8-10). Although algorithm 1 is tailored to the episodic setting, it can be easily extended to the infinite horizon setting.

4.2 Representation and transfer bias

Definition 4.4. *Given a distance function $d()$, transfer function f_{tr}, and constants $\boldsymbol{d_Q}$ and $\boldsymbol{d_{tr}}$, $\boldsymbol{\epsilon_c} \geq 0$ is the minimal non-negative constant satisfying $|B^\pi \tilde{Q}(x) - B^\pi \tilde{Q}(\bar{x})| \leq \boldsymbol{\epsilon_c} + d(x, \bar{x}, \boldsymbol{d_Q})$, $\forall (x, \bar{x}) : x_m = \bar{x}_m$, and for $\pi \in \{\tilde{\pi}^*, \pi^{\tilde{Q}}\}$, and*

$$\left| \int_{x'} p(x'|x, \pi)\Big(R(x, x'_s) + \gamma \tilde{Q}(x')\Big) - \int_{\bar{x}'} p(\bar{x}'|\bar{x}, \pi, \bar{x}'_s = f_{tr}(x, x'_s, \bar{x}))\Big(R(\bar{x}, \bar{x}'_s) + \gamma \tilde{Q}(\bar{x}')\Big) \right|$$
$$\leq \boldsymbol{\epsilon_c} + d(x, \bar{x}, \boldsymbol{d_{tr}}), \forall (x, x'_s, \bar{x}), \pi \in \{\tilde{\pi}^*, \pi^{\tilde{Q}}\}.$$

[2]This definition of a transfer function is specific to our algorithm, and is not meant to subsume similar concepts such as inter-task mappings [25].

Algorithm 1 PAC Transfer

```
1: Inputs: MDP $m$, horizon $H$, sample set $\tilde{S}$, number of neighbors $k$, transfer function $f_{tr}$, distance
   function $d()$, and constant $d_Q$.
2: Generate discretization set $\tilde{D}(\tilde{S}, m, d_Q)$, and set $\tilde{Q}(x) = 0 \; \forall x \in \tilde{D}(\tilde{S}, m, d_Q)$.
3: for $h = H - 1$ to $0$ do
4:     for $x \in \tilde{D}(\tilde{S}, m, d_Q)$ do
5:         $\tilde{Q}(x) = \frac{1}{k}\sum_{i=1}^{k}\left(\max\{0, \max_{a'_i}\{R(x, \bar{x}'_{i,s}) + \gamma\tilde{Q}(\bar{x}'_i) - d(x, x_i)\}\}\right)$ where $\bar{x}'_{i,a} = a'_i, \bar{x}'_{i,m} =$
           $m, \bar{x}'_{i,s} = f_{tr}(x_i, a'_i, x)$, and $(x_i, x'_{i,s})$ is the $i$-th sample returned by $nn_k(x, \tilde{S})$.
6:     end for
7: end for
8: for $h = 0$ to $H - 1$ do
9:     Perform action $\arg\max_{\bar{x}_a} \tilde{Q}(x_h)$, where $x_s$ is the state at step $h$ and $x_0$ is the starting state.
10: end for
11: function $nn_k(x, \tilde{S})$
12:     return the $k$ nearest samples to $x$ in $\tilde{S}$.
13: end function
14: function $\tilde{Q}(x)$
15: if $d(x, \tilde{D}(\tilde{S}, m, d_Q)) > d_Q$ then
16:     return 0.
17: else
18:     return $\tilde{Q}(\arg\min d(x, \tilde{D}(\tilde{S}, m, d_Q)))$ where ties are resolved by a deterministic function.
19: end if
20: end function
```

ϵ_c expresses the bias of algorithm 1. The first source of bias stems from the fact that algorithm 1 (or any other algorithm with finite computational requirements), can only represent functions of finite complexity. The slower the Bellman operator changes with respect to the chosen distance function, the lower the bias due to the fact that algorithm 1 is using a finite set of points to represent a continuous function. This is captured in the first part of definition 4.4.

The second source of bias is transfer. Transfer learning allows us to reduce sample complexity (variance) at the expense of introducing bias. When the transfer function is able to accurately model the dynamics in remote parts of the state-action-MDP space based on samples from other areas of the state-action-MDP space, transfer bias will be low. Conversely, when the transfer function is unable to accurately model the dynamics in one part of the state-action-MDP space based on samples from other areas of the state-action-MDP space, transfer bias will be high. This is captured in the second part of definition 4.4. The first integral is the (exact) Bellman operator for x applied to the value function produced by our algorithm, while the second integral is the Bellman operator for another state-action-MDP triple $\bar{x}$, but with the next state replaced by the value returned by the transfer function from x to $\bar{x}$.

When the bias introduced by transfer learning is unacceptably high, we have what is commonly referred to as negative transfer. Rather than thinking of negative transfer as a binary phenomenon (present/not present), definition 4.4 allows us to quantify its contribution. Unless the chosen transfer function is able to describe the changes in the dynamics across the state-action-MDP space perfectly (as would happen for example if the relative transfer function was used for transfer in a linear system), transfer learning will introduce some bias. Similarly, if the state-action-MDP space is continuous, using a finite set of points to represent the value function will introduce some bias. Given a fixed distance and transfer function, the extent of the bias can be managed by adjusting d_{tr} and d_Q.

Note that d_Q and d_{tr} are user defined constants (in other words the user decides the computational and sample complexity of the algorithm). Setting d_{tr} and d_Q needs to take into account policy performance as well as sample and computational complexity. As we will see in our analysis, the sample complexity of our algorithm is strongly dependent on d_{tr}, while the computational complexity is dependent on d_Q. The larger d_{tr} and d_Q are, the lower our algorithm's sample and computational complexity respectively. On the other hand, the performance of the policy produced by our algorithm is adversely affected by the bias ϵ_c. Definition 4.4 suggests that d_{tr} and d_Q should be set such that the two sources of bias are balanced.

While knowing the bias ϵ_c is not necessary for the execution of algorithm 1, our performance bounds depend on ϵ_c. A good choice of distance function will allow d_Q to be large without forcing ϵ_c to be large, allowing for good policy performance and low *computational* complexity. A good choice of transfer function will allow d_{tr} to be large without forcing ϵ_c to be large, allowing for good policy performance and low *sample* complexity.

4.3 PAC guarantees

Definition 4.5. *A* **policy cover $P_H^\pi(s, m, \epsilon)$** *is defined as a possibly infinite set of state-action pairs in MDP m such that the expected number of state-actions outside the set encountered in H steps when starting from state s and following policy π is bounded above by $\frac{\epsilon}{Q_{\max}}$.*

Theorem 4.6 below is the main theorem of this paper. Given a sample set of arbitrary distribution, it gives us a guarantee that as long as there exists a near-optimal policy for MDP m covered sufficiently well by the sample set, algorithm 1 will perform well in m with high probability. The samples can be samples collected from m, from MDPs similar to m, or any combination of the above.

Theorem 4.6. *Let s be the starting state for MDP m, and ϵ_c be defined as in definition 4.4. If $d(x, \bar{x}) \geq Q_{\max} \forall x, \bar{x} : x_h \neq \bar{x}_h$, $k \geq \frac{\hat{Q}^2_{\max}}{2\epsilon_s^2} \ln \frac{2|\tilde{D}(\tilde{S}, m, d_Q)|}{\delta}$, and there exists a policy $\tilde{\pi}^*$ with a policy cover $P_H^{\tilde{\pi}^*}(s, m, \epsilon_\pi)$ such that $V^{\tilde{\pi}^*}(s, m, 0) \geq V^*(s, m, 0) - \epsilon_\pi$, and $d_k(x, \tilde{S}) \leq d_{tr} \ \forall \ x \in P_H^{\tilde{\pi}^*}(s, m, \epsilon_\pi)$, then $V^{\pi^{\tilde{Q}}}(s, m, 0) > V^*(s, m, 0) - \epsilon_\pi - 2\xi\left(\epsilon_s + 2\epsilon_c\right)$ with probability at least $1 - \delta$, where $\pi^{\tilde{Q}}$ is the policy followed by algorithm 1, and $\xi = H$ if $\gamma = 1$ or $\xi = \min\left\{H, \frac{1}{1-\gamma}\right\}$ otherwise.*

The requirements for theorem 4.6 to hold are much more relaxed than the conditions of similar bounds [17]. No assumptions are made about the distribution of samples, the entire space for MDP m is not required to be well covered, and even the true optimal policy does not need to be covered. The fact that the computation of $\tilde{Q}$ is pessimistic ensures that as long as a policy with acceptable performance that is well covered exists, algorithm 1 will perform well with high probability. This is a departure from previous PAC algorithms that require a sufficient number of nearby samples for every state-action.

A brute force approach to ensuring that algorithm 1 will perform well with high probability on any MDP in the family of MDPs we are considering, is to guarantee that the entire space of the family is covered (has a sufficient number of samples within d_{tr} distance). Corollary 4.8 follows from theorem 4.6 and gives an upper bound on how many samples are sufficient in this scenario.

Definition 4.7. *The* **covering number $\mathcal{N}_{\mathcal{SAM}}(d_{tr})$** *of a state-action-MDP space is the cardinality of the largest minimal set C of state-action-MDP triples, such that for any x reachable from the starting state(s) of any $m \in M$, there exists $\bar{x} \in C$ such that $d(x, \bar{x}) \leq d_{tr}$.*

Corollary 4.8. *Let ϵ_c be defined as in definition 4.4, $d(x, \bar{x}) \geq Q_{\max} \forall x, \bar{x} : x_h \neq \bar{x}_h$, and $k = \frac{\hat{Q}^2_{\max}}{2\epsilon_s^2} \ln \frac{2|\tilde{D}(\tilde{S}, m, d_Q)|}{\delta}$. Given at least $\mathcal{N}_{\mathcal{SAM}}(d_{tr}) \frac{\hat{Q}^2_{\max}}{2\epsilon_s^2} \ln \frac{2|\tilde{D}(\tilde{S}, m, d_Q)|}{\delta}$ samples uniformly spaced across the state-action-MDP space, $V^{\pi^{\tilde{Q}}}(s, m, 0) > V^*(s, m, 0) - 2\xi\left(\epsilon_s + 2\epsilon_c\right)$ with probability $1 - \delta$ for any (s, m), where $\pi^{\tilde{Q}}$ is the policy followed by algorithm 1, and $\xi = H$ if $\gamma = 1$ or $\xi = \min\left\{H, \frac{1}{1-\gamma}\right\}$ otherwise.*

While the cardinality of the discretization set appears in our bounds, its influence is only logarithmic (which is exponentially smaller than the log-linear dependence of other PAC bounds). Corollary 4.8 is of interest for two reasons. The first is that it gives an upper bound on how many samples one would need to generate given a generative model. The second is that the sample complexity of PAC exploration algorithms scales with the number of uniformly spaced samples required before good performance can be guaranteed with high probability. Since $\mathcal{N}_{\mathcal{SAM}}(d_{tr})$ can be exponentially smaller in the dimensionality of the space than $\mathcal{N}_{\mathcal{SAM}}(d_Q)$, and algorithm 1 scales with $\mathcal{N}_{\mathcal{SAM}}(d_{tr})$ rather than $\mathcal{N}_{\mathcal{SAM}}(d_Q)$, the same improvement can be achieved in the exploration setting.

5 Discussion

The sample complexity of modern PAC RL algorithms is log-linear with respect to the size of the space. In continuous or discrete spaces for which a distance metric exists, the effective size of the space is strongly dependent on the desired quality of approximation. The size of the space is expressed by the covering number, the number of hyperballs required to completely cover the state-action-MDP space (see definition 4.7). Better approximation quality requires hyperballs of smaller radius. As the radius of each hyperball decreases, the covering number can grow exponentially to the dimensionality of the space. In effect, the sample complexity of PAC algorithms for spaces with a distance metric grows exponentially to the dimensionality of the space. Algorithm 1 can lead to exponential sample complexity reduction with respect to the dimensionality of the state-action-MDP space. The majority of existing PAC RL algorithms make no distinction between the radius of the hyperball where each sample can be used to compute the value function, and the radius of the hyperballs where the value function has constant (or smoothly interpolated) value [16, 7, 18, 14]. Theorem 4.6 tells us that as long as a transfer function is available that is able to accurately model the dynamics in remote parts of the state-action-MDP space based on samples from other areas of the state-action-MDP space the radius of the hyperball where each sample can be (re)used can be much larger than the radius of constant value hyperballs typically used in PAC algorithms. This can lead to an exponential reduction in sample complexity with respect to the dimensionality of the state-action-MDP space.

5.1 The advantage of pessimism

PAC exploration algorithms compute an estimate of the value function that is optimistic with respect to uncertainty, while most batch mode algorithms compute a maximum likelihood estimate. Optimistic and maximum likelihood estimates can result in the overestimation of the value of poorly covered suboptimal policies. As a result, even if the value of a well performing policy can be accurately estimated from a sample set, optimistic and maximum likelihood estimates can result in the selection of a poorly covered suboptimal policy.

In many real-world situations such as when samples have been collected from expert demonstrations, one can have access to samples from a high quality policy that does not cover the entire state-action space. If the expert acts near-optimally, it is likely that their demonstrations will only visit a subset of the state-action space. Low value state-actions may never be visited. Note that the "expert" in this case does not need to be human. For example we may interested in taking over control of cooling a datacenter from classical control algorithms, in order to improve energy efficiency [6]. Given the safety constraints of such a system it is unlikely that the entire state-action space will be covered with samples, yet given that the existing policy has acceptable performance, an acceptable policy is guaranteed to be covered by the sample set. Additionally, in many domains prior knowledge can allow us to deduce that a significant portion of the state-action space would not be visited by a near-optimal policy, and thus does not need to be sampled. Pessimism in the face of uncertainty gives algorithm 1 a significant advantage in these situations, by ensuring that the value of poorly covered suboptimal policies will not be overestimated (thus those suboptimal policies will not be selected). Rather than requiring the entire state-action space to be well covered with samples, algorithm 1 only requires that there exists an adequately covered near-optimal policy. To the best of our knowledge, algorithm 1 is the first algorithm for which the existence of an adequately covered near-optimal policy is sufficient to guarantee approximately optimal performance with high probability.

5.2 Selecting distance and transfer functions

Similarly to how feature based algorithms require a set of features, kernel based algorithms require a kernel function, and deep learning algorithms require an appropriate architecture, distance based algorithms require an appropriate distance function. Considerable progress has been made over the past few decades, and automatic discovery of distance metrics is a promising area of research [21, 26] with deep autoencoders [8] achieving impressive results on areas ranging from raw images [9], to speech and game playing.

While the focus of this paper is on a theoretical analysis of algorithm 1 given a distance and transfer function, we can find examples in the literature of simple distance and transfer functions that work very well in practice. Previous work on PAC reinforcement learning that does not use a transfer function but imposes smoothness constraints on the value function across an entire MDP family [18] is implicitly using the identity transfer function and $d_{tr} = d_Q$. Brunskill et al. [2] present CORL, a PAC algorithm for intra-task transfer in continuous state, discrete action MDPs, where transitions can be described as an offset from the current state plus Gaussian noise. States are classified into types, with the parameters of the dynamics being the same for all states in the same type. In effect, CORL is using the relative transfer function, and a distance function that returns 0 for states of the same type and infinity for states of different types. The authors present experiments with a real robot navigating a multi-surface environment, and show that a camera can be used to successfully classify the surface into different types. Since the algorithm by Brunskill et al. arises as a special case of our algorithm, the experimental results in that paper can be replicated by our algorithm.

While the local linearity assumption made by the relative transfer function is relatively uncommon in the reinforcement learning literature, it is a common assumption in control systems. It has been used in practical applications ranging from temperature and voltage regulation, to autopilot systems for recreational drones and commercial aircraft. Nevertheless, there exist domains for which improving on the relative transfer function requires only minimal domain specific knowledge, such as for example domains with saturating or quantized dynamics.

6 Related work

The work of Brunskill et al. [2] described in the previous section is the only existing PAC algorithm for intra-task transfer. Our work generalizes that idea to arbitrary transfer functions and noise, as well as to inter-task transfer. Additionally, algorithm 1 enjoys significantly better dependence on the covering number and planning horizon.

Mann and Choe [15] introduce weak admissible heuristics, and show how they can be used for transfer learning. Weak admissible heuristics transfer values rather than samples, which makes them orthogonal to our approach. It would be straightforward to construct an algorithm that takes advantage of both a transfer function and a weak admissible heuristic. The concept of a transfer function is related to inter-task mappings [25], a type of transfer function for inter-domain value function transfer.

While transfer learning learning has attracted significant attention from the reinforcement learning community (see for example the survey by Taylor and Stone [24]) only a handful of algorithms exist for which theoretical guarantees have been proven. Lazaric and Restelli [12] present three transfer learning algorithms (AST, BAT, and BTT), and prove performance bounds for AST and BAT. All three algorithms assume that: 1) There is a number of source tasks and a single target task. 2) We have access to generative models for the environments. 3) The target task can be expressed accurately enough as a linear combination of the source tasks. AST uses all samples collected from the source tasks to learn a value function for the "average MDP". Using a generative model of the target task, BAT tries to find the best proportions from which to generate samples from the source tasks. Finally BTT, for which no theoretical guarantees are known, allows constraints to be placed on how many samples can be generated from each task. These algorithms could be useful in a scenario where evaluating the target task is significantly more computationally expensive than evaluating the source tasks (as long as the target task can also be approximated accurately enough as a linear combination of the source tasks).

Brunskill and Li [3], as well as Liu, Guo and Brunskill [14] focus on the identification task for the discrete and continuous settings respectively. Compared to naively starting exploration from scratch every time a task is sampled, both algorithms are shown to offer an improvement in the aggregate sample complexity when run for a finite number of tasks. This improvement is achieved because finding which cluster a task belongs to requires fewer samples than exploring in that task from scratch. However, transfer learning is not used to reduce the sample complexity of the *learning* phase. The sample complexity of the learning phase is the number of clusters times the sample complexity of learning in each cluster from scratch. The algorithms presented in those papers could be combined with the approach presented in this paper, achieving the best of both worlds.

The fact that pessimism in the face of uncertainty can be advantageous has been previously identified by the robust MDP literature [20]. Our work identified a new advantage of pessimism, and contributed a more computationally efficient algorithm. While robust MDP algorithms require solving multiple linear programs at every iteration, algorithm 1 is no more computationally expensive than maximum likelihood value iteration.

References

[1] M. G. Azar, R. Munos, and H.J. Kappen. Minimax PAC-bounds on the sample complexity of reinforcement learning with a generative model. *Machine Learning Journal*, 91(3):325–349, 2013.

[2] Emma Brunskill, Bethany R. Leffler, Lihong Li, Michael L. Littman, and Nicholas Roy. CORL: A continuous-state offset-dynamics reinforcement learner. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2008.

[3] Emma Brunskill and Lihong Li. Sample complexity of multi-task reinforcement learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2013.

[4] Bruno Castro Da Silva and Andrew G. Barto. TD-$\Delta\pi$: A model-free algorithm for efficient exploration. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12, pages 886–892. AAAI Press, 2012.

[5] T. Dietterich, M. Taleghan, and M Crowley. PAC optimal planning for invasive species management: Improved exploration for reinforcement learning from simulator-defined mdps. In *AAAI Conference on Artificial Intelligence*, July 2013.

[6] Richard Evans and Jim Gao. DeepMind AI reduces google data centre cooling bill by 40%.

[7] Robert C. Grande, Thomas J. Walsh, and Jonathan P. How. Sample efficient reinforcement learning with Gaussian processes. In *International Conference on Machine Learning*, pages 1332–1340, June 2014.

[8] Geoffrey Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[10] Tor Lattimore and Marcus Hutter. PAC bounds for discounted MDPs. In *Proceedings of the 23th International Conference on Algorithmic Learning Theory*, volume 7568 of *Lecture Notes in Computer Science*, pages 320–334. Springer Berlin / Heidelberg, 2012.

[11] A. Lazaric, M. Restelli, and A. Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the Twenty-Fifth Annual International Conference on Machine Learning (ICML-2008)*, pages 544–551, Helsinki, Finalnd, July 2008.

[12] Alessandro Lazaric and Marcello Restelli. Transfer from multiple mdps. In *Advances in Neural Information Processing Systems 24*, pages 1746–1754, 2011.

[13] Lihong Li. *A unifying framework for computational reinforcement learning theory*. PhD thesis, Rutgers University, New Brunswick, NJ, USA, 2009.

[14] Yao Liu, Zhaohan Guo, and Emma Brunskill. PAC continuous state online multitask reinforcement learning with identification. In *Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2016.

[15] Timothy A. Mann and Yoonsuck Choe. Directed exploration in reinforcement learning with transferred knowledge. *JMLR Workshop and Conference Proceedings: EWRL*, 24:59–76, 2012.

[16] Jason Pazis and Ronald Parr. PAC optimal exploration in continuous space Markov decision processes. In *AAAI Conference on Artificial Intelligence*, pages 774–781, July 2013.

[17] Jason Pazis and Ronald Parr. Sample complexity and performance bounds for non-parametric approximate linear programming. In *AAAI Conference on Artificial Intelligence*, July 2013.

[18] Jason Pazis and Ronald Parr. Efficient PAC-optimal exploration in concurrent, continuous state MDPs with delayed updates. In *AAAI Conference on Artificial Intelligence*, February 2016.

[19] Jason Pazis, Ronald E Parr, and Jonathan P How. Improving PAC exploration using the median of means. In *Advances in Neural Information Processing Systems 29*, pages 3898–3906. 2016.

[20] Marek Petrik and Dharmashankar Subramanian. Raam: The benefits of robustness in approximating aggregated MDPs in reinforcement learning. In *Advances in Neural Information Processing Systems 27*, pages 1979–1987. Curran Associates, Inc., 2014.

[21] Bernhard Schőlkopf. The kernel trick for distances. *Advances in Neural Information Processing Systems*, 1993.

[22] Alexander L. Strehl, Lihong Li, and Michael L. Littman. Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research*, 10:2413–2444, December 2009.

[23] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, Massachusetts, 1998.

[24] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.

[25] Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007.

[26] Eric P. Xing, Andrew Y. Ng, Michael I. Jordan, and Stuart Russell. Distance metric learning, with application to clustering with side-information. *Advances in Neural Information Processing Systems*, pages 505–512, 2003.

7 Analysis

We will begin our analysis by extending the theory of Bellman error MDPs [18] to the episodic setting. Bellman error MDPs will allow us to prove bounds that depend on how well some policy $\tilde{\pi}^*$ is covered, rather than requiring all state-action-MDP triples to be covered with samples. Once we have extended Bellman error MDPs to the episodic setting, we will show that due to the pessimistic nature of algorithm 1, the Bellman error for the policy followed by the algorithm will not be too positive for any point in the discretization set. We also show that the Bellman error for some policy $\tilde{\pi}^*$ will not be too negative for any point in the discretization set. We then extend the proof of the above to any state-action-MDP triple. Finally, we use these properties to show that as long as certain conditions are met, algorithm 1 will perform well with high probability.

Let m be an MDP $(\mathcal{S}, \mathcal{A}, P, R, \gamma, H)$ with Bellman operator B^{π} for policy π, and Q an approximate value function for m. Additionally, let $0 \leq Q^{\pi}(x) \leq Q_{\max}\ \forall x, \pi$. Let the Bellman error MDP $\boldsymbol{m_{\epsilon(\pi,Q)}}$ be an MDP which differs from m only in its reward function which is defined as $\boldsymbol{R_{\epsilon(\pi,Q)}(x)} = Q(x) - B^{\pi}Q(x)$ ($Q(x)$ and B^{π} are the approximate value function and Bellman operator of the original MDP m). We will use $\boldsymbol{B^{\pi}_{\epsilon(\pi,Q)}}$ to denote $m_{\epsilon(\pi,Q)}$'s Bellman operator under policy π, and $\boldsymbol{Q^{\pi}_{\epsilon(\pi,Q)}}$ as well as $\boldsymbol{V^{\pi}_{\epsilon(\pi,Q)}}$ to denote its value function.

Most results on Bellman error MDPs follow directly from theorem 7.1 and basic properties of MDPs.

Theorem 7.1. *The return of π over m is equal to Q minus the return of π over $m_{\epsilon(\pi,Q)}$[3]:*

$$Q^{\pi}(x) = Q(x) - Q^{\pi}_{\epsilon(\pi,Q)}(x)\ \forall(x).$$

Proof. We will use induction to prove our claim, starting with $x_h = H - 1$ as our base case: From the definition of the Bellman error MDP we have that:

$$\begin{aligned} Q^{\pi}_{\epsilon(\pi,Q)}(x|x_h = H-1) =& Q(x,|x_h = H-1) - B^{\pi}Q(x,|x_h = H-1) \\ =& Q(x,|x_h = H-1) - Q^{\pi}(x,|x_h = H-1). \end{aligned}$$

Let $Q^{\pi}(x,h) = Q(x,h) - Q^{\pi}_{\epsilon(\pi,Q)}(x,h)\ \forall x$ hold for some $x_h \geq 1$ we will prove that this implies it also holds for $x'_h = x_h - 1$:

$$\begin{aligned} & Q^{\pi}_{\epsilon(\pi,Q)}(x|x_h = h-1) \\ &\quad = Q(x|x_h = h-1) - B^{\pi}Q(x|x_h = h-1) + \gamma \int_{x'} P(x'|x,\pi) Q^{\pi}_{\epsilon(\pi,Q)}(x'|x'_h = h) \\ &\quad = Q(x|x_h = h-1) - \int_{x'} P(x'|x,\pi)\left(R(x,x'_s) - \gamma Q(x'|x'_h = h)\right) \\ &\qquad\qquad + \gamma \int_{x'} P(x'|x,\pi)\left(Q(x'|x'_h = h) - Q^{\pi}(x'|x'_h = h)\right) \\ &\quad = Q(x|x_h = h-1) - \int_{x'} P(x'|x,\pi)\left(R(x,x'_s) + \gamma Q^{\pi}(x'|x'_h = h)\right) \\ &\quad = Q(x|x_h = h-1) - Q^{\pi}(x|x_h = h-1). \end{aligned}$$

□

Corollary 7.2 bounds the range of the Bellman error MDP value function, a property that will prove very useful in the analysis of our algorithm. It follows from Theorem 7.1 and the fact that $0 \leq Q^{\pi}(x,h) \leq Q_{\max}$.

Corollary 7.2. *Let $0 \leq Q(x) \leq Q_{\max}\ \forall(x)$. Then:*

$$-Q_{\max} \leq Q^{\pi}_{\epsilon(\pi,Q)}(x) \leq Q_{\max}\ \forall(x,\pi).$$

Lemma 7.3 below (a consequence of theorem 7.1), proves that the difference between the expected, discounted reward of an approximately optimal and the greedy policy over Q is bounded above by the inverse difference of the value of those policies in their respective Bellman error MDPs.

[3] Note that this is true for any policy π not just the greedy policy over Q.

Lemma 7.3. *$\forall(s, m, h, Q)$:*

$$V^{\tilde{\pi}^*}(s,m,h) - V^{\pi^Q}(s,m,h) \leq V^{\pi^Q}_{\epsilon(\pi^Q,Q)}(s,m,h) - V^{\tilde{\pi}^*}_{\epsilon(\tilde{\pi}^*,Q)}(s,m,h)$$

Proof. Let $x_s = s$, $x_m = m$, and $x_h = h$.

$$\begin{aligned}
&\int_x P(x|\tilde{\pi}^*)Q(x) \leq \int_x P(x|\pi^Q)Q(x) \Rightarrow \\
&\int_x P(x|\tilde{\pi}^*)\left(Q^{\tilde{\pi}^*}(x) + Q^{\tilde{\pi}^*}_{\epsilon(\tilde{\pi}^*,Q)}(x)\right) \\
&\qquad \leq \int_x P(x|\pi^Q)\left(Q^{\pi^Q}(x) + Q^{\pi^Q}_{\epsilon(\pi^Q,Q)}(x)\right) \Rightarrow \\
&\int_x P(x|\tilde{\pi}^*)Q^{\tilde{\pi}^*}(x) - \int_x P(x|\pi^Q)Q^{\pi^Q}(x) \leq \\
&\qquad \int_x P(x|\pi^Q)Q^{\pi^Q}_{\epsilon(\pi^Q,Q)}(x) - \int_x P(x|\tilde{\pi}^*)Q^{\tilde{\pi}^*}_{\epsilon(\tilde{\pi}^*,Q)}(x) \Rightarrow \\
&V^{\tilde{\pi}^*}(s,m,h) - V^{\pi^Q}(s,m,h) \leq V^{\pi^Q}_{\epsilon(\pi^Q,Q)}(s,m,h) - V^{\tilde{\pi}^*}_{\epsilon(\tilde{\pi}^*,Q)}(s,m,h).
\end{aligned}$$

□

Lemma 7.4. *Let $P^{\pi}_{H}(s, m, \epsilon_\pi)$ be a policy cover for π and $\int_x P(x|\pi)\left(Q(x) - B^{\pi}Q(x)\right) > -\epsilon_0 \, \forall(x) \in P^{\pi}_{H}(s, m, \epsilon_\pi)$. Then*

$$V^{\pi}_{\epsilon(\pi,Q)}(s,m,h) \geq -\xi\epsilon_0 - \epsilon_\pi$$

where $\xi = H$ if $\gamma = 1$ or $\xi = \min\left\{H, \frac{1}{1-\gamma}\right\}$ otherwise.

Proof. The expected reward in the Bellman error MDP is bounded below by $-\epsilon_0$ for all state-actions in the policy cover. Since the expected number of state-actions outside the policy cover encountered in H steps is bounded above by $\frac{\epsilon_\pi}{Q_{\max}}$, and $\int_x P(x|\pi)\left(Q(x) - B^{\pi}Q(x)\right) \geq -Q_{\max}$, we have that $V^{\pi}_{\epsilon(\pi,Q)}(s,m,h) \geq -\xi\epsilon_0 - \epsilon_\pi$. □

Definition 7.5. *The approximate pessimistic Bellman operator is defined as*

$$\tilde{B}^{\pi}\tilde{Q}(x) = \frac{\sum_{i=1}^{k}\left(\max\{0, R(x, \bar{x}'_{i,s}) + \gamma\tilde{Q}(\bar{x}'_i) - d(x, x_i)\}\right)}{k}$$

where $(x_i, x'_{i,s})$ is the i-th sample returned by $nn_k(x, \tilde{S})$, $\bar{x}'_{i,s} = f_{tr}(x_i, x'_{i,s}, x)$, $\bar{x}'_{i,a} = \pi(\bar{x}'_{i,s})$, and $\bar{x}'_{i,m} = x_m$.

Lemma 7.6 below bounds the probability that there exists an element in the discretization set with Bellman error of unacceptably high magnitude.

Lemma 7.6. *Let $\tilde{Q}$ be the value function produced by algorithm 1, and $k \geq \frac{\hat{Q}^2_{\max}}{2\epsilon_s^2}\ln\frac{2|\tilde{D}(\tilde{S},m,d_Q)|}{\delta}$. The probability that there exists at least one $x \in \tilde{D}(\tilde{S}, m, d_Q)$ such that*

$$\tilde{Q}(x) - B^{\pi^{\tilde{Q}}}\tilde{Q}(x) \geq \epsilon_s + \epsilon_c$$

or

$$\tilde{Q}(x) - B^{\tilde{\pi}^*}\tilde{Q}(x) \leq -\epsilon_s - \epsilon_c - 2d_k(x, \tilde{S}, d_{tr})$$

is bounded above by δ.

Proof. Let Y be the set of k samples used to compute $\tilde{B}^{\pi}\tilde{Q}(x)$ for a fixed $(\tilde{Q}, x)$ and define $f^{\pi}(z_1, \ldots z_k) = \tilde{B}^{\pi}\tilde{Q}(x)$, where $z_1, \ldots z_k$ are realizations of independent (from the Markov property) variables whose outcomes are possible next states, one for each sample in Y. The outcomes of the variables (which is where the Markov property ensures independence) are the next states the transitions lead to, not the state-action-MDP triples the samples originate from. The state-action-MDP triples the samples originate from are fixed with respect to f, and no assumptions are made about their distribution. Additionally, $\tilde{Q}(x)$ is fixed with respect to f (we are examining the effects of a single application of $\tilde{B}^{\pi}\tilde{Q}(x)$ to the fixed value $\tilde{Q}(x)$, while varying the next-states that samples in Y land on). Then:

$$\sup_{z_1,\dots z_k,\hat{z}_i} |f(z_1,\dots z_k) - f(z_1,\dots,z_{i-1}\hat{z}_i,z_{i+1}\dots z_k)| = c_i \leq \frac{\hat{Q}_{max}}{k},$$

and $\sum_{i=1}^{k}(c_i)^2 \leq k\frac{\hat{Q}_{max}^2}{k^2} = \frac{\hat{Q}_{max}^2}{k}$. From McDiarmid's inequality we have

$$\begin{aligned} &P\left(\tilde{B}^{\pi^{\tilde{Q}}}\tilde{Q}(x) - \mathrm{E}\left[\tilde{B}^{\pi^{\tilde{Q}}}\tilde{Q}(x)\right] \geq \epsilon_s\right) \\ &= P\left(f^{\pi^{\tilde{Q}}}(z_1,\dots z_k) - \mathrm{E}\left[f^{\pi^{\tilde{Q}}}(z_1,\dots z_k)\right] \geq \epsilon_s\right) \\ &\leq e^{-\frac{2\epsilon_s^2}{\Sigma_{i=1}^k(c_i^2)}} \leq e^{-\frac{2\epsilon_s^2 k}{\hat{Q}_{max}^2}} \leq \frac{\delta}{2|\tilde{D}(\tilde{S},m,d_Q)|}, \end{aligned}$$

and

$$\begin{aligned} &P\left(\tilde{B}^{\tilde{\pi}^*}\tilde{Q}(x) - \mathrm{E}\left[\tilde{B}^{\tilde{\pi}^*}\tilde{Q}(x)\right] \leq -\epsilon_s\right) \\ &= P\left(f^{\tilde{\pi}^*}(z_1,\dots z_k) - \mathrm{E}\left[f^{\tilde{\pi}^*}(z_1,\dots z_k)\right] \leq -\epsilon_s\right) \\ &\leq e^{-\frac{2\epsilon_s^2}{\Sigma_{i=1}^k(c_i^2)}} \leq e^{-\frac{2\epsilon_s^2 k}{\hat{Q}_{max}^2}} \leq \frac{\delta}{2|\tilde{D}(\tilde{S},m,d_Q)|}. \end{aligned}$$

From definition 4.4 we have that $B^{\pi}\tilde{Q}(x) - \epsilon_c - 2d_k(x,\tilde{S},d_{tr}) \leq \mathrm{E}\left[\tilde{B}^{\pi}\tilde{Q}(x)\right] \leq B^{\pi}\tilde{Q}(x) + \epsilon_c$. From the definition of our algorithm we have that $\tilde{B}^{\tilde{\pi}^*}\tilde{Q}(x) \leq \tilde{B}^{\pi^{\tilde{Q}}}\tilde{Q}(x) = \tilde{Q}(x)$. Substituting above we have that for a fixed x where $x \in \tilde{D}(\tilde{S},m,d_Q)$

$$P\left(\tilde{Q}(x) - B^{\pi^{\tilde{Q}}}\tilde{Q}(x) \geq \epsilon_s + \epsilon_c\right) \leq \frac{\delta}{2|\tilde{D}(\tilde{S},m,d_Q)|},$$

and

$$\begin{aligned} &P\left(\tilde{Q}(x) - B^{\tilde{\pi}^*}\tilde{Q}(x) \leq -\epsilon_s - \epsilon_c - 2d_k(x,\tilde{S},d_{tr})\right) \\ &\qquad \leq \frac{\delta}{2|\tilde{D}(\tilde{S},m,d_Q)|}. \end{aligned}$$

Taking a union bound over all $x \in \tilde{D}(\tilde{S},m,d_Q)$ completes our proof. □

Lemma 7.7 extends the result from lemma 7.6 to the entire set of state-actions.

Lemma 7.7. *Let $\tilde{Q}$ be the value function produced by algorithm 1, and $k \geq \frac{\hat{Q}_{max}^2}{2\epsilon_s^2}\ln\frac{2|\tilde{D}(\tilde{S},m,d_Q)|}{\delta}$. The probability that there exists at least one x such that*

$$\tilde{Q}(x) - B^{\pi^{\tilde{Q}}}\tilde{Q}(x) \geq \epsilon_s + 2\epsilon_c,$$

or at least one x such that $d(x,\tilde{D}(\tilde{S},m,d_Q)) \leq d_Q$ and

$$\tilde{Q}(x) - B^{\tilde{\pi}^*}\tilde{Q}(x) \leq -\epsilon_s - 2\epsilon_c - 2d_k(x,\tilde{S},d_{tr})$$

is bounded above by δ.

Proof. If $d(x, \tilde{D}(\tilde{S}, m, d_Q)) > d_Q$ then $\tilde{Q}(x) = 0$. Since $B^{\pi^{\tilde{Q}}}\tilde{Q}(x) \geq 0$ we have that $\tilde{Q}(x) - B^{\pi^{\tilde{Q}}}\tilde{Q}(x) \leq 0 < \epsilon_s + 2\epsilon_c$. Let $\bar{x} = \arg d(x, \tilde{D}(\tilde{S}, m, d_Q))$. Then

$$\begin{aligned}\tilde{Q}(x) - B^{\pi^{\tilde{Q}}}\tilde{Q}(x) &= \tilde{Q}(\bar{x}) - B^{\pi^{\tilde{Q}}}\tilde{Q}(x)\\ &\leq \tilde{Q}(\bar{x}) - B^{\pi^{\tilde{Q}}}\tilde{Q}(\bar{x}) + \epsilon_c,\end{aligned}$$

and if $d(x, \tilde{D}(\tilde{S}, m, d_Q)) \leq d_Q$

$$\begin{aligned}\tilde{Q}(x) - B^{\tilde{\pi}^*}\tilde{Q}(x) &= \tilde{Q}(\bar{x}) - B^{\tilde{\pi}^*}\tilde{Q}(x)\\ &\geq \tilde{Q}(\bar{x}) - B^{\tilde{\pi}^*}\tilde{Q}(\bar{x}) - \epsilon_c,\end{aligned}$$

where in both inequalities we used the definition of $\tilde{Q}$ and definition 4.4. Substituting into lemma 7.6 concludes our proof. □

7.1 Proof of theorem 4.6

Proof. From lemma 7.7 and the fact that $d_k(x, \tilde{S}) \leq d_{tr}\ \forall\ x \in P_H^{\tilde{\pi}^*}(s, m, \epsilon_\pi)$, we have that with probability at least $1 - \delta$

$$\tilde{Q}(x) - B^{\pi^{\tilde{Q}}}\tilde{Q}(x) < \epsilon_s + 2\epsilon_c\ \forall x,$$

and

$$\tilde{Q}(x) - B^{\tilde{\pi}^*}\tilde{Q}(x) > -\epsilon_s - 2\epsilon_c\ \forall (x) \in P_H^{\tilde{\pi}^*}(s, m, \epsilon_\pi).$$

It follows that $\int_x P(x|\pi^{\tilde{Q}})Q^{\pi^{\tilde{Q}}}_{\epsilon(\pi^{\tilde{Q}},Q)}(x) < \xi(\epsilon_s + 2\epsilon_c)$ and from lemma 7.4 we have that $\int_x P(x|\tilde{\pi}^*)Q^{\tilde{\pi}^*}_{\epsilon(\tilde{\pi}^*,Q)}(x) \geq -\xi(\epsilon_s + 2\epsilon_c) - \epsilon_\pi$. Substituting into lemma 7.3 we have

$$\begin{aligned}&V^{\pi^{\tilde{Q}}}(s, m, 0) - V^*(s, m, 0)\\ &= \int_x P(x|\pi^*, x_h = 0)Q^{\pi^*}(x) - \int_x P(x|\pi^{\tilde{Q}}, x_h = 0)Q^{\pi^{\tilde{Q}}}(x)\\ &\leq \int_x P(x|\tilde{\pi}^*, x_h = 0)Q^{\tilde{\pi}^*}(x) - \int_x P(x|\pi^{\tilde{Q}}, x_h = 0)Q^{\pi^{\tilde{Q}}}(x) + \epsilon_\pi\\ &\leq \int_x P(x|\pi^{\tilde{Q}}, x_h = 0)Q^{\pi^{\tilde{Q}}}_{\epsilon(\pi^{\tilde{Q}},Q)}(x) - \int_x P(x|\tilde{\pi}^*, x_h = 0)Q^{\tilde{\pi}^*}_{\epsilon(\tilde{\pi}^*,Q)}(x) + \epsilon_\pi\\ &< \xi(\epsilon_s + 2\epsilon_c) + \xi(\epsilon_s + 2\epsilon_c) + \epsilon_\pi\\ &= 2\xi(\epsilon_s + 2\epsilon_c) + \epsilon_\pi.\end{aligned}$$

□

APPENDIX D Pac Intra-task Knowledge Transfer in MDPs

PAC intra-task Knowledge Transfer in MDP

Wuming Zhang

May 2017

1 Abstract

Probably approximately correct (PAC) algorithms for Reinforcement Learning has shown great theoretical guarantees. However, the large sample complexity has been the bottleneck of its practical applications. Transfer learning, on the other hand, can potentially reduce the sample complexity. In this project, we introduce the intra-task transfer learning into the PAC reinforcement learning algorithms. We demonstrate the sample complexity reduction from a pendulum experiment.

2 Introduction

In the last few years, PAC reinforcement learning has gained some popularity for its nice theoretical guarantee that, with high probability, the algorithm will produce a policy that performs almost as well as the optimal policy. The required sample size is guaranteed to be at most polynomial to the parameters of the problem. Such parameters include 1) the size of the state-action-MDP space, 2) the planning horizon, 3) the tolerance of the deviation from the optimal policy, and 4) the probability of failure.

Previous works showed that providing PAC guarantees requires large sample complexity in many cases [1] [2] [3]. In RL, the state-action MDP space contributes the most to the sample complexity. In this project, we want to reduce the state-action MDP space while maintaining the PAC guarantees by introducing an intra-task transfer function. We want to show that in practice, this new PAC transfer algorithm can indeed reduce sample complexity.

3 Method

3.1 Notations

3.1.1 MDP

Markov Decison Process family consists of state space S, action space A, MDP family M, Markov transition model P, reward function R, discount factor γ, and horizon time H. $\mathbf{Q}^{\pi}(\mathbf{x}, \mathbf{h})$ is defined as the expected, accumulated, discounted reward from x at step h, when all decisions from step $h+1$ follow policy π.

3.1.2 Sample Set

Sample set $\tilde{\mathbf{S}}$ is defined as the set of all samples retained by our algorithm.

3.1.3 Distance Function

$\mathbf{d}(\mathbf{x}, \bar{\mathbf{x}})$ is defined to be the distance of two different state-action-MDPs in some well-defined distance metric.

3.1.4 Cover Set

The cover set $\tilde{\mathbf{C}}(\tilde{\mathbf{S}}, \mathbf{m})$ is defined as the portion of the state-action space where at least one sample is less than Q_{max}.

3.1.5 Intra-task transfer

A transfer function $\mathbf{f_{tr}}(\mathbf{x}, \mathbf{x'_s}, \bar{\mathbf{x}}) \rightarrow \bar{x'_s}$ is defined as function that takes in a state-action-MDP triple, an associated next state, and a target state-action-MDP triple, and predicts the next state of the target state-action-MDP triple.

3.1.6 Discretization Set

A discretization set $\tilde{\mathbf{D}}(\tilde{\mathbf{S}}, \mathbf{m}, \mathbf{d_Q})$ is defined as a discretization of the cover set $\tilde{\mathbf{C}}(\tilde{\mathbf{S}}, \mathbf{m})$, such that for every $x \in \tilde{\mathbf{C}}(\tilde{\mathbf{S}}, \mathbf{m})$, there exist $\bar{x} \in \tilde{\mathbf{D}}(\mathbf{S}, \mathbf{m}, \mathbf{d_Q})$, such that $\mathbf{d}(\mathbf{x}, \bar{\mathbf{x}}) \leq d_Q$

3.2 PAC Transfer Algorithm

The PAC Transfer Algorithm is shown below. It consists of two main stages. The first stage is to calculate the value function Q using the sample set and the discretization set, along with the transfer function. The second stage is to use the calculated Q to apply policy.

4 Experiment

4.1 Setups

We consider the following pendulum problem. Suppose we have a pendulum hanging downwards. We want to apply forces to its attached cart so that we can bring it as close as the balanced position as possible. The state space then consists of the vertical angle θ and the angular velocity $\dot{\theta}$. The action space is $\{-50, 0, 50\}$. Uniform noise in $[-10, 10]$ is also added to the actions. The reward is set to be 1 if $|\theta| < \pi/2$; it is set to be 0 otherwise. A discount factor

Algorithm 1 PAC Transfer Algorithm

```
1: Inputs: MDP $m$, horizon $H$, sample set $\tilde{S}$, number of neighbors $k$, transfer
   function $f_{tr}$, distance function $d()$, and constant $d_Q$
2: Generate discretization set $\mathbf{\tilde{D}(\tilde{S}, m, d_Q)}$
3: Set $\tilde{Q}_{cur}(x) = 0, \tilde{Q}_{prev}(x) = 0, \forall x \in \mathbf{\tilde{D}(\tilde{S}, m, d_Q)}$
4:
5: for h = 1:H do
6:     for $x \in \mathbf{\tilde{D}(S, m, d_Q)}$ do
7:         $\tilde{Q}_{cur}(x) = \frac{1}{k}\sum_{i=1}^{k}(max\left\{0, max_{a'_i}\left\{R(x, \bar{x}'_{i,s}, h) + \gamma\tilde{Q}_{prev}(\bar{x'_i}) - d(x, x_i))\right\}\right\})$
8:         where $\bar{x}'_{i,a} = a'_i, \bar{x}'_{i,m} = m, \bar{x}'_{i,s} = f_{tr}(x_i, x'_{i,s}, x)$, and $(x_i, x'_{i,s})$ is
   the $i_{th}$ sample returned by $NN_k(x, \tilde{S})$
9:     Set $\tilde{Q}_{prev}(x) = \tilde{Q}_{cur}(x)$
10:
11: while terminal condition unsatisfied do
12:     Perform $action = argmax_{\bar{x_a}}\tilde{Q}_{cur}(x)$
13:
14: procedure NNK$(x, \tilde{S})$
15:     return the $k$ nearest samples to $x$ in $\tilde{S}$
```

of 0.9 is used. We assumed that the absolute magnitude of the angular velocity will not exceed 3.

4.2 Generation of the Sample Set and Discretization Set

Since we have a 2-dimensional state space, $(\theta, \dot{\theta})$, we want to sample a Discretization Set of size $m \times n$. We first evenly divide the state space into a $m \times n$ grid, so that $cell_{ij}$ represents $(\theta, \dot{\theta})$, where]

$$\theta \in [-\pi/2 + (i-1) \times \pi/m, -\pi/2 + i \times \pi/m]$$

$$\dot{\theta} \in [-3 + (j-1) \times 6/n, -3 + j \times 6/n]$$

We first construct the sample set S by sampling t pairs of $(\theta, \dot{\theta})$ from each cell we just defined. We then construct the discretization set by randomly taking one of the sampling point from each cell. In this way, for each of the three actions, we can generate a sample set of size $m \times n \times t$, and a corresponding discretization set of size $m \times n$.

4.3 Model Detail

We set the distance function to be the norm-2 distance of the state vector. Note that we set the distance to infinity for those with different actions. We chose the horizon to be 20 since this experiment does not require much long term planning. We used $2t$ nearest neighbors in our model.

4.4 Transfer Function Specification

We first did a sanity check using the identity transfer function. That is, $\mathbf{f}_{\mathbf{tr}}(\mathbf{x}, \mathbf{x}'_s, \bar{\mathbf{x}}) \rightarrow \mathbf{x}'_s$. We then applied our PAC transfer algorithm on a more complicated transfer function, that is symmetric transfer function. Intuitively, if a person can perform well on the pendulum balancing task at a state $(\theta_0, \dot{\theta}_0)$, such skill should be able to transfer to perform well at a new state $(-\theta_0, -\dot{\theta}_0)$. If we believe that $\bar{x}$ is close to $-x$, then the symmetric transfer function is, $\mathbf{f}_{\mathbf{tr}}(\mathbf{x}, \mathbf{x}'_s, \bar{\mathbf{x}}) \rightarrow -\mathbf{x}'_s$. The possible existence of symmetry can be checked in the NN_k function in the algorithm by changing the method of distance calculation. We can then simply add an indicator variable specifying whether symmetry exists. Such indicator variable is also returned in the NN_k function.

4.5 Identity Transfer vs. Symmetric Transfer

To quantitatively compare the result from an identity transfer and a symmetric transfer. We varied the number of samples that Sample Set sampled from each cell, that is, t. For different values of t, we recorded the total number of iterations the experiment lasted before encountering a failure ($|\theta| > \pi/2$). For computation simplicity, we cap the maximum of iteration to be $20,000$. We repeat the experiment for ten times. The results can be shown in Table 1 and Fig. 1.

5 Result and Conclusion

From the result, we can observe that, with a fixed samples per cell in the sample set, PAC Transfer algorithm with symmetric transfer function tends to last a lot more iterations then with a simple identity transfer function, the variance of the symmetric transfer function result also more quickly converges to zero since there is the $20,000$ max iteration.

The results match our expectation since with a symmetric transfer function, the number of samples required to achieve similar performance should be roughly cut into half.

A example of the heatmap of the learned Q matrix is shown in Fig. 2. The corresponding best policy is shown in Fig. 3. Note that these two plots are just for one scenario where we sample 3 pairs per cell with the identity transfer function.

Above all, We can conclude that the PAC transfer learning algorithms can indeed reduce sample complexity in some practical applications.

Sample per cell	1	2	3	4	5
Identity Transfer	51	493	5,782	16,045	20,000
Symmetric Transfer	380	12,531	20,000	20,000	20,000

Table 1: Identity vs. Symmetric

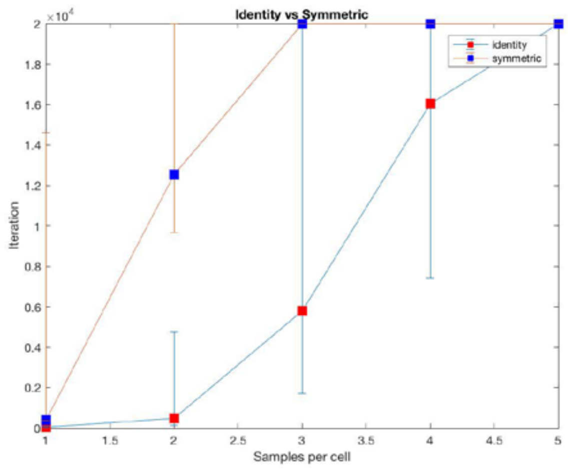


Figure 1: Identity vs. Symmetric Results with Error Bars

6 Future Work

Even though this project shows the practical application of the PAC Transfer learning algorithms for RL, the experiments we conducted were still fairly simple. To show that this can be applied to more real-world problems, we need to design some more complicated experiments. Some examples related to pendulum experiment include introducing some transfer function for different pendulum mass and length.

Furthermore, although transfer function, if given, can be very powerful in terms of reducing the sample complexity, defining different transfer functions for every problem seems to be unrealistic. How to learn a general purpose transfer function that can apply to various problems is something worth further study on.

References

[1] Mohammad Gheshlaghi Azar, Rémi Munos, and Hilbert J. Kappen. Minimax PAC bounds on the sample complexity of reinforcement learning with a generative model. *Machine Learning*, 91(3):325–349, 2013.

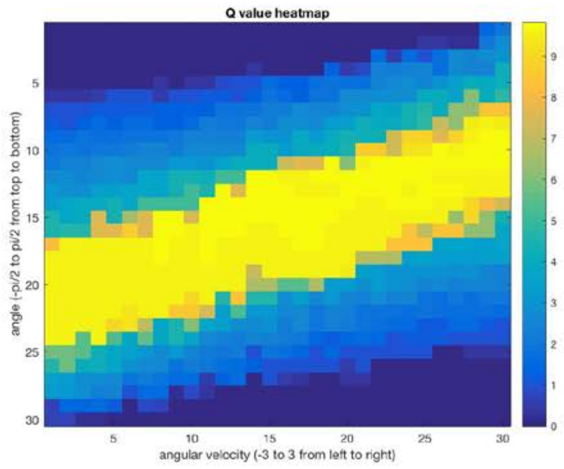


Figure 2: A heatmap of the learned Q matrix with 3 samples/cells, identity transfer function

[2] Tor Lattimore and Marcus Hutter. PAC bounds for discounted MDPs. *CoRR*, abs/1202.3890, 2012.

[3] Alexander L. Strehl, Lihong Li, and Michael L. Littman. Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research*, 10:2413–2444, 2009.

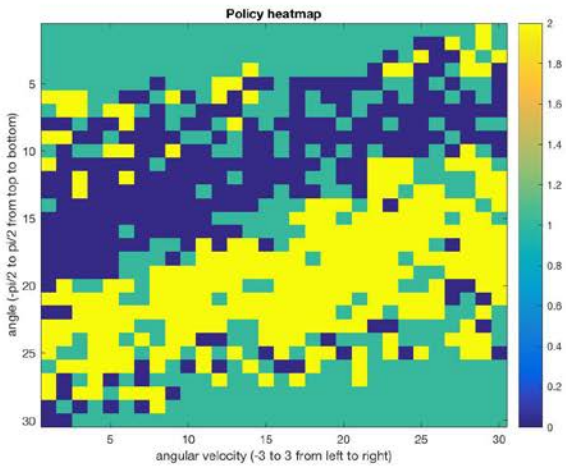


Figure 3: A heatmap of the best policy corresponding to Fig. 2

APPENDIX E Transfer with GAN in Policy Gradient Reinforcement Learning

Transfer Learning with Generative Adversarial Network in Policy Gradient Reinforcement Learning

Will Wang
Department of Computer Science
Duke University
Durham, NC 27708
`ww109@duke.edu`

Abstract

Reinforcement learning is concerned with how agents ought to take actions in an environment so as to maximize cumulative reward. Transfer learning, on the other hand, typically refers to attempts to decrease training time by learning a source task before learning the target task. There have been many approaches to do transfer learning within the reinforcement learning domain. In this paper, we present a new method of doing so by using Generative Adversarial Networks.

1 Introduction

In reinforcement learning, transferring knowledge gained from tasks solved earlier to solve a new target task, can help, either in terms of speeding up the learning process or in terms of achieving better final performance. Various methods have been proposed to do transfer given know a relationship between tasks. Other work, however, focuses on using different tools to autonomously discover such a relationship and then do transfer. Due to the hardness of this problem, it remains a big challenge to effectively learn good relationship between tasks.

In this paper, we proposed a novel method to use Generative Adversarial Networks (GAN) to solve this challenge. The GAN framework is a deep learning model recently introduced by Goodfellow, et al. (2014). It can draw samples with arbitrary distribution by letting a generator and a discriminator to play a minimax game. It could also model a conditional distribution and thus help to make huge progress in domain adaption problems. Inspired by GAN's success in domain adaption problems, we aim to learn a mapping between states in two tasks in reinforcement learning and then transfer the policy accordingly.

2 Related Work

In this section, we briefly review advances in transfer reinforcement learning and Generative Adversarial Networks so that it will be more natural to introduce our proposed method.

2.1 Transfer Reinforcement Learning

As pointed out by Taylor, et al. (2009), different transfer algorithms have different problem settings and make different assumptions. However, they generally share following steps: 1. Given a target task, select an appropriate source task(s) to transfer from. 2. Learn how the source task and target are related. 3. Effectively transfer knowledge from the source task(s) to the target task. One option to provide the relationship between the source task and target task is by inter-task mapping. For example, such mapping can be decomposed into action-mapping and state-variable mapping between two tasks. From the beginning of 2000, much work focuses on cases where inter-task mapping is available. In other words, they assume the second step is provided by a human. More recently, much work also tries to learn a mapping between two tasks autonomously. Usually, one type of alignment method needs to be used to find a good mapping. After the mapping is provided, there are a handful of ways to do transfer. For example, we can use the Q-values learned from the source task to initiate the Q-values in the target task. The other option is to directly transfer without mapping. In deep reinforcement learning, for instance, we can transfer the weights of neural network to solve the new task. This can be viewed as a way to transfer the features. Using some more advanced algorithms, we can also transfer the policy as well, and then fine-tune that.

2.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) have led to significant improvements in image generation Radford, et al. (2015). The basic idea of GANs is to simultaneously train a discriminator and a generator. The discriminator is trained to distinguish real samples of a dataset from fake samples produced by the generator. The generator uses input from an easy-to-sample random source, and is trained to produce fake samples that the discriminator cannot distinguish from real data samples. From a game theory point of a view, the convergence of a GAN is reached when the generator and the discriminator reach a Nash equilibrium.

There are two threads of GAN research moving on quickly. One is to make GANs more stable and powerful. Wasserstein GAN (WGAN) [6], Energy Based GAN (EBGAN) [7] and Boundary Equilibrium GAN (BEGAN) [8] are recent noteworthy examples on this track. The other track is to apply GANs to other domains and achieve state of the art results. Recent work on domain transfer is a great example. Almost concurrently, DiscoGAN [9] and CycleGAN [10] produce astonishing results in image to image translation task. They can unsupervisedly discover paired relationships between instances from two datasets surprisingly well. Even though state space is very different from image space, this makes us think if we can apply GANs to do transfer in Reinforcement Learning domains.

3 Method

3.1 Problem Formulation

Before we introduce our method, we first formally define our problem. Reinforcement Learning problems are typically formulated as a Markov Decision Process (MDP) $M =< S, A, T, r >$, S is the set of states, A is the set of actions that agents could execute, $T : S \times A \times S \rightarrow [0, 1]$ is a state transition probability function specifying the task dynamics and $r : S \times A \times S \rightarrow R$. A policy $\pi : S \times A \rightarrow [0, 1]$ is defined as a conditional probability over actions given state. The goal for an agent is to sequentially choose actions to maximize its expected return during interaction with environment.

Our transfer problem considers a source domain with MDP $M_S =< S_S, A_S, T_S, r_S >$ and a target domain with MDP $M_T =< S_T, A_T, T_T, r_T >$. In general, they can have different state spaces, action spaces, dynamics and reward functions. One way to transfer knowledge from source to target is by mapping optimal <state, action, next state> pair into state and action spaces in the target domain. To do so, one must provide an inter-task mapping χ to project such a triple from source to target. Such χ can be decomposed into two sub-mappings: an inter-state mapping χ_S and inter-action mapping χ_A.

3.2 Alignment with GAN

In our study, we want to leverage GAN's strong modeling ability to learn a potentially very complicated inter-task mapping. The basic assumption we make is that the dynamics in the source domain resembles the dynamics in the target domain. Besides, the reward function should also share some structural similarity. We also assume the inter-action mapping χ_A is provided and only desire to learn the inter-state mapping χ_S.

In order to learn an inter-task mapping, we need examples from both domains. We obtain trajectories of states S_S in the source task and trajectories of states S_T in the target task by applying random policy in both domains. Because we assume actions are already aligned, we extract state-next state pairs from both domains as $[(S_S^{(1)}, S_S^{(2)})]$ and $[(S_T^{(1)}, S_T^{(2)})]$ respectively.

We define a function $G(S)$ as a neural network to project states from source domain to states target domain. It takes a state from the source task domain as input and outputs a state in the target task domain. This is the generator in the GAN framework and is implemented by a multilayer perceptron. We also define a second multilayer perceptron $D((S^{(1)}, S^{(2)}))$ that outputs a single scaler. $D(x)$ represents the probability that the input pair comes from the target domain state-next state pair distribution rather than from generator G. We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize $log(1 - D(G(S_S^{(1)}), G(S_S^{(2)})))$.

In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

$$min_G max_D V(D, G) = E[\log D((S_T^{(1)}, S_T^{(2)})] + E[\log(1 - D(G(S_S^{(1)}), G(S_S^{(2)})))] \quad (1)$$

The intuition here is that the generator has to find a projection so that the the state-next state distribution is aligned. Because the generator is only defined on the state space, and the transition dynamics are similar, we should expect this produces nice alignment across two domains.

3.3 Using Alignment for Knowledge Transfer

After we learn the alignment projector G, we can use knowledge from the source task to guide the training in the target task. We first use a policy gradient method to learn an optimal policy π_S in the source domain. To apply transfer, we first initialize π_T by training only on the transfer reward $r_{transfer}$.

$$r_{transfer} = -||\pi_S(S_S) - \pi_T(G(S_S))||_2 \quad (2)$$

This term forces the target policy to output the same action as the source policy in the corresponding source state. Again, here we assume actions are already properly aligned between source and target tasks. We ran a standard policy gradient method on this reward to initialize the policy. After that, we train the target policy on the real target reward without the transfer reward.

The whole algorithm is summarized in the following:

Algorithm 1 Transfer by GAN

```
1: procedure LEARN PROJECTOR
2:     Sample pairs $(S_S^{(1)}, S_S^{(2)})$ from source task, and pairs $(S_S^{(1)}, S_S^{(2)})$ from target task both from random policies.
3:     Learn projector $G$ by training a GAN with equation (1) as objective using above samples
4: procedure TRANSFER INITIALIZE POLICY
5:     Sample $m$ source states $S_S$ with optimal policy $\pi_S^{(*)}$
6:     Project $S_S$ to target domain with projector $G$
7:     Use policy gradient to train on transfer reward according to equation (2)
8:     Yield initialized target task policy $\pi_T^{(0)}$
9: procedure IMPROVE POLICY
10:    Start with $\pi_T^{(0)}$ and then train on real reward
11:    Return optimal target policy $\pi_T^{(*)}$
```

4 Experiment Result

We test our transfer algorithm on the Cart Pole environment as a proof of concept experiment. Here, we are trying to transfer knowledge between Cart Pole with different parameters.

The goal of Cart Pole (Figure 1) is to swing up and then balance the pole vertically. The system dynamics are described via a four-dimensional state vector <x,x',θ,θ'>, respectively representing the position, velocity of the cart, angle and angular velocity of the pole. Actions are to put a +1 or -1 force to the cart. Notice that the actions of Cart Pole with different parameter are naturally aligned.

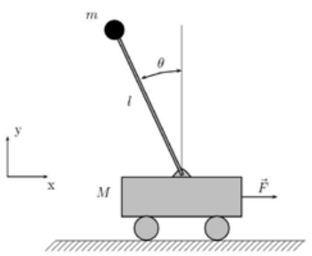


Figure 1: Cart Pole Environment.

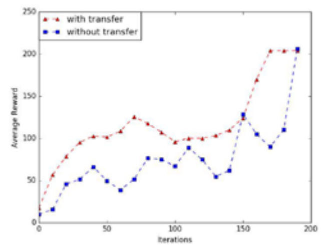


Figure 2: Transfer Result.

Figure 2 shows average target task reward over iterations with and without transfer. It's plotted after policy is initialized by transfer for the case with transfer. This transfer policy initialization phase only trained with 50 optimal trajectories with 1 iterations. We can see that transfer-initialized policy outperforms standard policy gradient trained from scratch. It has much better initialized performance and converges faster. This demonstrates that our algorithm is capable of providing helpful target policy initialization.

We applied a few tricks in implementation of our model. Because the training of GANs is well-known to be unstable, we add an additional penalizing term $||G(S) - S||_2$ to encourage the transformed states to be close to original input to avoid bad initialization by random projection. We gradually decay the weight of this term during training so that the distribution could be matched better. Also we select result from saved training checkpoints based on how close the projected next state from the actual next state in the target task.

5 Conclusion

We introduced a technique for autonomous transfer with policy gradient reinforcement learning. Our approach employs Generative Adversarial Networks to generate an inter-task mapping, which is then used to transfer source knowledge to the target domain on the Cart-Pole environment. We demonstrate its effectiveness on cart-pole environment, showing it's capable of improving the agent's initial performance and convergence speed.

References

[1] Taylor, M. E., Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. Journal of Machine Learning Research, 10(Jul), 1633-1685.

[2] Ammar, H. B., Eaton, E., Ruvolo, P., Taylor, M. E. (2015, January). Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment. In Proc. of AAAI.

[3] Gupta, A., Devin, C., Liu, Y., Abbeel, P., Levine, S. (2017). Learning invariant feature spaces to transfer skills with reinforcement learning. arXiv preprint arXiv:1703.02949.

[4] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).

[5] Radford, A., Metz, L., Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

[6] Qi, G. J. (2017). Loss-Sensitive Generative Adversarial Networks on Lipschitz Densities. arXiv preprint arXiv:1701.06264.

[7] Zhao, J., Mathieu, M., LeCun, Y. (2016). Energy-based generative adversarial network. arXiv preprint arXiv:1609.03126.

[8] Berthelot, D., Schumm, T., Metz, L. (2017). BEGAN: Boundary Equilibrium Generative Adversarial Networks. arXiv preprint arXiv:1703.10717.

[9] Jun-Yan Zhu*, Taesung Park*, Phillip Isola, and Alexei A. Efros. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", in arxiv, 2017.

[10] Kim, T., Cha, M., Kim, H., Lee, J., Kim, J. (2017). Learning to Discover Cross-Domain Relations with Generative Adversarial Networks. arXiv preprint arXiv:1703.05192.

APPENDIX F PAC Reinforcement Learning Knowledge Transfer via Manifold Alignment

PAC Reinfocement Learning Knowledge Transfer via Manifold Alignment

Wuming Zhang

August 2017

1 Abstract

Probably approximately correct (PAC) algorithms for Reinforcement Learning has shown great theoretical guarantees. However, the large sample complexity has been the bottleneck of its practical applications. Manifold alignment, on the other hand, can effectively transfer knowledge, and potentially reduce the sample complexity. In this project, we introduce the intra-task transfer learning into the PAC reinforcement learning algorithms. We explore sample transfer through manifold alignment. We demonstrate the sample complexity reduction for several classic control experiments with a perfect transfer function and a learned transfer function.

2 Introduction

In the last few years, PAC reinforcement learning has gained some popularity for its nice theoretical guarantee that, with high probability, the algorithm will produce a policy that performs almost as well as the optimal policy. The required sample size is guaranteed to be at most polynomial to the parameters of the problem. Such parameters include 1) the size of the state-action-MDP space, 2) the planning horizon, 3) the tolerance of the deviation from the optimal policy, and 4) the probability of failure.

Previous works showed that providing PAC guarantees requires large sample complexity in many cases [2] [3] [4]. In RL, the state-action-MDP space contributes the most to the sample complexity. It is natural to explore the possibility of knowledge transfer to reduce sample complexity. One idea is that samples in source domain and target domain may have some connections in their underlying manifold structures, and manifold alignment is introduced to align their manifolds to achieve knowledge transfer [5]. A special case of such technique, unsupervised manifold alignment, is shown to be effective in inter-task transfer in the context of policy gradient RL [1]. In this project, we first introduce a PAC Transfer Algorithm, in which we demonstrate that an intra-task transfer function can reduce the state-action MDP sample complexity while maintaining the PAC guarantees. We then show that manifold alignment can be used to learn such transfer function and effectively reduce sample complexity in the context of the PAC transfer algorithm.

3 Method

3.1 PAC Transfer Algorithm

3.1.1 Notations

MDP: Markov Decison Process family consists of state space S, action space A, MDP family M, Markov transition model P, reward function R, discount factor γ, and horizon time H. $\mathbf{Q}^{\pi}(\mathbf{x}, \mathbf{h})$ is defined as the expected, accumulated, discounted reward from x at step h, when all decisions from step $h+1$ follow policy π.
Sample Set: Sample set $\tilde{\mathbf{S}}$ is defined as the set of all samples retained by our algorithm.
Distance Function: $\mathbf{d}(\mathbf{x}, \tilde{\mathbf{x}})$ is defined to be the distance of two different state-action-MDPs in some well-defined distance metric.
Cover Set: The cover set $\tilde{\mathbf{C}}(\tilde{\mathbf{S}}, \mathbf{m})$ is defined as the portion of the state-action space where at least one sample is less than Q_{max}.
Intra-task transfer: A transfer function $\mathbf{f_{tr}}(\mathbf{x}, \mathbf{x}'_{\mathbf{s}}, \bar{\mathbf{x}}) \to \bar{x}'_s$ is defined as function that takes in a state-action-MDP triple, an associated next state, and a target state-action-MDP triple, and predicts the next state of the target state-action-MDP triple.
Discretization Set: A discretization set $\tilde{\mathbf{D}}(\tilde{\mathbf{S}}, \mathbf{m}, \mathbf{d_Q})$ is defined as a discretization of the cover set $\tilde{\mathbf{C}}(\tilde{\mathbf{S}}, \mathbf{m})$, such that for every $x \in \tilde{\mathbf{C}}(\tilde{\mathbf{S}}, \mathbf{m})$, there exist $\bar{x} \in \tilde{\mathbf{D}}(\mathbf{S}, \mathbf{m}, \mathbf{d_Q})$, such that $\mathbf{d}(\mathbf{x}, \bar{\mathbf{x}}) \leq d_Q$

3.1.2 Detailed Algorithm

The PAC Transfer Algorithm is shown below. It consists of three main steps. The first step is to derive a sample set S and a discretization set D. The second step is to calculate the value function Q using the sample set and the discretization set, along with the transfer function. The last step is to use the calculated Q to apply policy.

3.2 Manifold Alignment

3.2.1 Problem Definition

We first want to define the problem in the context of RL state mapping. More specifically, let X_s be a sample set of state from the source domain, where $\mathcal{X}_s$ is the manifold of X_s. Let X_t be a sample set of state from the target domain, where $\mathcal{X}_t$ is the manifold of X_t. We have partial knowledge about their correspondence, $x_s^i \in X_s \leftrightarrow x_t^i \in X_t$. We want to map X_s and X_t to a new space, modeling their correspondence, and preserving the local structure of

Algorithm 1 PAC Transfer Algorithm

```
1:  Inputs: MDP $m$, horizon $H$, sample set $\tilde{S}$, number of neighbors $k$, transfer
    function $f_{tr}$, distance function $d()$, and constant $d_Q$
2:  Generate discretization set $\tilde{\mathbf{D}}(\tilde{\mathbf{S}}, \mathbf{m}, \mathbf{d_Q})$
3:  Set $\tilde{Q}_{cur}(x) = 0, \tilde{Q}_{prev}(x) = 0, \forall x \in \tilde{\mathbf{D}}(\tilde{\mathbf{S}}, \mathbf{m}, \mathbf{d_Q})$
4:
5:  for h = 1:H do
6:      for $x \in \tilde{\mathbf{D}}(\mathbf{S}, \mathbf{m}, \mathbf{d_Q})$ do
7:          $\tilde{Q}_{cur}(x) = \frac{1}{k}\sum_{i=1}^{k}(max\left\{0, max_{a'_i}\left\{R(x, \bar{x}'_{i,s}, h) + \gamma\tilde{Q}_{prev}(\bar{x'_i}) - d(x, x_i))\right\}\right\})$
8:          where $\bar{x}'_{i,a} = a'_i, \bar{x}'_{i,m} = m, \bar{x}'_{i,s} = f_{tr}(x_i, x'_{i,s}, x)$, and $(x_i, x'_{i,s})$ is
    the $i_{th}$ sample returned by $NN_k(x, \tilde{S})$
9:      Set $\tilde{Q}_{prev}(x) = \tilde{Q}_{cur}(x)$
10:
11: while terminal condition unsatisfied do
12:     Perform $action = argmax_{\bar{x}_a}\tilde{Q}_{cur}(x)$
13:
14: procedure NNK$(x, \tilde{S})$
15:     return the $k$ nearest samples to $x$ in $\tilde{S}$
```

each set at the same time. In our specific context, we choose to use feature-level alignment since it is easier to generalize to new instance and better fits our application.

3.2.2 Notations

X_s: $p_s \times m_s$ matrix, representing source domain sample set
X_t: $p_t \times m_t$ matrix, representing target domain sample set
W_s: $m_s \times m_s$ relationship matrix of the source domain sample set, $W_s^{i,j}$ is the similarity of x_s^i and x_s^j (heat kernel), $W_s^{i,j} = e^{-dist(x_s^i, x_s^j)/\delta}$
W_t: $m_t \times m_t$ relationship matrix of the target domain sample set, $W_t^{i,j}$ is the similarity of x_t^i and x_t^j, $W_t^{i,j} = e^{-dist(x_t^i, x_t^j)/\delta}$
D_s: $m_s \times m_s$ diagonal matrix, where $D_s^{i,j} = \sum_j W_s^{i,j}$
D_t: $m_t \times m_t$ diagonal matrix, where $D_t^{i,j} = \sum_j W_t^{i,j}$
L_s: $L_s = D_s - W_s$
L_t: $L_t = D_t - W_t$
$W_{s,t}$: $m_s \times m_t$ correspondence matrix. $W_{s,t}^{i,j} = 1$ if x_s^i corresponds with x_t^j; otherwise, $W_{s,t}^{i,j} = 0$.
Ω_s: $m_s \times m_s$ diagonal matrix, $\Omega_s(i,i) = \sum_j W_{s,t}^{i,j}$
Ω_t: $m_t \times m_t$ diagonal matrix, $\Omega_t(i,i) = \sum_i W_{s,t}^{i,j}$
Z: $Z = \begin{pmatrix} X_s & 0 \\ 0 & X_t \end{pmatrix}$

D: $D = \begin{pmatrix} D_s & 0 \\ 0 & D_t \end{pmatrix}$
L: $L = \begin{pmatrix} L_s + \mu\Omega_s & -\mu W_{s,t} \\ -\mu W_{s,t} & L_t + \mu\Omega_t \end{pmatrix}$
F_s: a $p_s \times d$ matrix that maps each x_s^i to a new d-dimensional space.
F_t: a $p_t \times d$ matrix that maps each x_t^i to a new d-dimensional space.
γ: $\gamma = (F_s^T, F_t^T)$

3.2.3 Cost function

The cost function is defined as the following:
$C(F_s, F_t) = \mu \sum_{i=1}^{m_s} \sum_{j=1}^{m_t} \left\| F_s^T x_s^i - F_t^T x_t^j \right\|^2 W_{s,t}^{i,j}$
$+0.5 \sum_{i=1}^{m_s} \sum_{j=1}^{m_s} \left\| F_s^T x_s^i - F_s^T x_s^j \right\|^2 W_s^{i,j} + 0.5 \sum_{i=1}^{m_t} \sum_{j=1}^{m_t} \left\| F_t^T x_t^i - F_t^T x_t^j \right\|^2 W_t^{i,j}$
This cost function represents a trade-off between preserving local geometry of each data set and capturing their correspondence.

3.2.4 Manifold Alignment Algorithm [5]

- Construct matrices W_s and W_t to capture the local geometry of each data set. Construct matrix $W_{s,t}$ to model the correspondence between the two data sets.
- Compute L, Z and D as indicated above to model the joint structure of the two data sets.
- Derive the optimal mapping function using d minimum eigenvectors, $\gamma_1, \gamma_2, ..., \gamma_d$ of the generalized eigenvalue decomposition $ZLZ^T\gamma = \lambda ZDZ^T\lambda$
- Let F_s be part of $[\gamma_1, \gamma_2, ..., \gamma_d]$ from row 1 to row p_s. Let F_t be part of $[\gamma_1, \gamma_2, ..., \gamma_d]$ from row 1 to row p_t. Then $F_s^T x_s^i$ and $F_t^T x_t^j$ are in the same space and can be aligned.

3.2.5 Discussion on Correspondence

Full Correspondence: If the correspondence matrix $W_{s,t}$ captures full correspondence information of the two data sets, then the problem is called supervised manifold alignment.
Partial Correspondence: If the correspondence matrix $W_{s,t}$ captures partial correspondence information of the two data sets, then the problem is called semi-supervised manifold alignment.
No Correspondence: If there is no correspondence information at all, then the problem is called unsupervised manifold alignment. Since x_s^i and x_t^j cannot be directly compared, we can instead use x_s^i and its neighbors to capture the local geometry of x_s^i. We can characterize the local geometry of x_t^j in a similar way [6]. Then we can compare the local relations to find correspondence and thus generating the correspondence matrix $W_{s,t}$.

4 Experiment

4.1 Pendulum

We consider the following pendulum balance problem. Suppose we have a pendulum hanging upwards. We want to apply forces to its attached cart so that we can bring it as close as the balanced position as possible. The state space then consists of the vertical angle θ and the angular velocity $\dot{\theta}$. We assumed that the angle is in a range of $[-\pi/2, \pi/2]$, and the angular velocity is in a range of $[-3, 3]$. The action space is $\{-50, 0, 50\}$. Uniform noise in $[-10, 10]$ is also added to the actions. The reward is set to be 1 if $|\theta| < \pi/2$; it is set to be 0 otherwise. A discount factor of 0.9 is used.
Since we have a 2-dimensional state space, $(\theta, \dot{\theta})$, we want to sample a discretization set of size $m \times n$ (we choose $m = 30$, $n = 30$). We first evenly divide the state space into a $m \times n$ grid, so that $cell_{ij}$ represents $(\theta, \dot{\theta})$, where

$$\theta \in [-\pi/2 + (i-1) \times \pi/m, -\pi/2 + i \times \pi/m]$$

$$\dot{\theta} \in [-3 + (j-1) \times 6/n, -3 + j \times 6/n]$$

We first construct the sample set S by sampling t pairs of $(\theta, \dot{\theta})$ from each cell we just defined. We then construct the discretization set by randomly taking one of the sampling point from each cell. In this way, for each of the three actions, we can generate a sample set of size $m \times n \times t$, and a corresponding discretization set of size $m \times n$.
We set the distance function to be the norm-2 distance of the state vector. Note that we set the distance to infinity for those with different actions. We choose the horizon $H = 20$ and use $k = 10$ nearest neighbors in our model.

4.2 Pendulum Swing Up

Similar to the Pendulum setup, now we have a pendulum hanging downwards. We want to apply forces to its attached cart to bring it up to a balanced position as quickly as possible. We assume that the angle is in a range of $[-\pi, \pi]$, and the angular velocity is in a range of $[-10, 10]$. The discretization set is of size 20×60. The terminal state is any of the four cells in the center of the discretization set (both θ and $\dot{\theta}$ are close to zero). The reward is set to be 1 if the pendulum reaches the terminal state and 0 otherwise. All the rest of the settings are set to be the same as the pendulum experiment.

4.3 Transfer Function Specification

4.3.1 Identity Transfer

We first conduct a sanity check using the identity transfer function. That is, $\mathbf{f_{tr}}(\mathbf{x}, \mathbf{x}'_s, \bar{\mathbf{x}}) \rightarrow \mathbf{x}'_s$. This is just to demonstrate the baseline performance of the PAC RL algorithm when a transfer function is not involved.

4.3.2 Symmetry Transfer

We then applied our PAC transfer algorithm on a slightly more complicated transfer function, the symmetry transfer function. Intuitively, if a person can perform well on the pendulum balancing task at a state $(\theta_0, \dot{\theta}_0)$, such skill should be able to transfer to perform well at a new state $(-\theta_0, -\dot{\theta}_0)$. The symmetric transfer function is formally defined as, $\mathbf{f_{tr}}(\mathbf{x}, \mathbf{x'_s}, \bar{\mathbf{x}}) \rightarrow -\mathbf{x'_s}$, if $\bar{x}$ and x are symmetric. The possible existence of symmetry can be checked in the NN_k function in the algorithm by changing the method of distance calculation. We can then simply add an indicator variable specifying whether symmetry exists. Such indicator variable is also returned in the NN_k function.

4.3.3 Semi-supervised Manifold Alignment

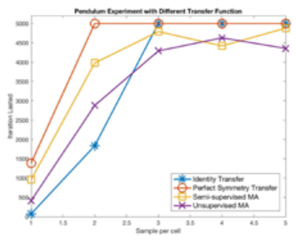
We then use semi-supervised manifold alignment technique to learn such a symmetry transfer function. We assume that we have perfect knowledge about the action transfer. We randomly select 20% of the source domain state sample as the target domain state sample. We then label the correspondence of 80% of the target domain state sample. The sensitivity to the amount of target domain data and correspondence pairs is discussed in the result section. The transfer function is formally defined as, $\mathbf{f_{tr}}(\mathbf{x}, \mathbf{x'_s}, \bar{\mathbf{x}}) \rightarrow \bar{\mathbf{x}}'_{\mathbf{s}}$. We choose $\delta = 1$ in the heat kernel of the relationship matrix W_s and W_t. We use $\mu = 2$ in the manifold alignment cost function. We also set the dimension of the new space to be 2 so that the mapping matrix F_s and F_t can be square matrices to avoid reconstruction error. Now for a new instance in source domain x_s, we can compute its corresponding instance x_t in the target domain, $x_t = F_t^{-1} F_s x_s$.

4.3.4 Unsupervised Manifold Alignment

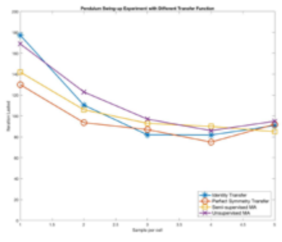
For unsupervised manifold alignment, again we assume that we have perfect knowledge about the action transfer. We select 70% of the source domain state sample as the target domain state sample. We then label the correspondence of 80% of the target domain state sample. The sensitivity to the amount of target domain data and correspondence pairs is discussed in the result section. We use $k = 10$ nearest neighbour to generate the correspondence matrix $W_{s,t}$. All the rest of the parameters are kept the same as semi-supervised manifold alignment.

5 Result and Discussion

To quantitatively compare the difference between the transfer functions mentioned above, we vary the number of samples in each cell of the sample set S for both experiments. We run the experiments 20 times for each number of samples per cell, and record the median number of iterations the experiment lasts before it terminates. Note that for both experiments, we set the maximum of iteration to be $5,000$. The results are shown in Fig. 1.



(a) Pendulum Experiment

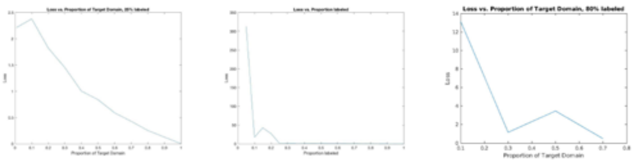


(b) Pendulum Swing-up Experiment

Figure 1: Comparison of Different Transfer Functions

We can observe that, with a fixed number of sample per cell in the sample set, PAC Transfer algorithm with a perfect symmetric transfer function tends to perform better than with a simple identity transfer function. A transfer function learned via semi-supervised manifold alignment generally performs slight worse than the perfect transfer function but still better than the identity transfer function. For unsupervised manifold alignment, it is much more difficult to learn such a transfer function, and its learning performance has quite large variance. It generally perform similarly to the identity transfer function and sometimes could be worse. We also observe a lot of variations between runs with a fixed sample number per cell, for all four transfer functions. This is mostly caused by the noise that we add in our simulation.

We also test the influence of the amount of target domain data and correspondence information to the performance of both semi-supervised and unsupervised manifold alignment (see Fig. 2). With a given source domain data set X_s (we use samples from the pendulum experiment), we use $-X_s$ as the groundtruth label of the symmetry transfer function. For both manifold alignment methods, two mappings, F_s and F_t are obtained. We apply these two mappings to all source domain data X_s. We derive the testing result $X_{test} = F_t^{-1} F_s X_s$. The loss function is simply defined as the Euclidean distance between each instance of X_{test} and $-X_s$. We use the average loss to evaluate the performance. We can observe that for both manifold alignment algorithms, more data always provide better performance. We notice that semi-supervised manifold alignment can performance decently well with a fairly small amount of data, but on the other hand, unsupervised manifold alignment requires much more data to reach similar performance. We also observe that more correspondence information has a positive impact to the performance of the semi-supervised manifold alignment.



(a) The Influence of the Amount of Target Domain Data on Semi-supervised Manifold Alignment

(b) The Influence of the Amount of Correspondence on Semi-supervised Manifold Alignment

(c) The Influence of the Amount of Target Domain Data on Unsupervised Manifold Alignment

Figure 2: Sensitivity Test on Manifold Alignment

6 Future Work

We can conclude that the introduction of a transfer function to PAC RL algorithm can indeed reduce sample complexity in some practical applications. Manifold alignment shows some promises in learning such transfer function.

However, the experiments we conduct are still fairly simple. To show that this can be applied to more real-world problems, we need to design some more complicated experiments. It would also be interesting to explore more on transfer learning without correspondence information since our experiments show that the unsupervised manifold alignment method is somewhat unreliable. Another problem is that the PAC algorithms still does not scale well even with some plausible transfer function. Some further studies on reducing sample complexity, such as learning through demonstration, would be helpful.

References

[1] Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew E Taylor. Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment. 2015.

[2] Mohammad Gheshlaghi Azar, Rémi Munos, and Hilbert J. Kappen. Minimax PAC bounds on the sample complexity of reinforcement learning with a generative model. *Machine Learning*, 91(3):325–349, 2013.

[3] Tor Lattimore and Marcus Hutter. PAC bounds for discounted MDPs. *CoRR*, abs/1202.3890, 2012.

[4] Alexander L. Strehl, Lihong Li, and Michael L. Littman. Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research*, 10:2413–2444, 2009.

[5] Chang Wang and Sridhar Mahadevan. A general framework for manifold alignment. 2009.

[6] Chang Wang and Sridhar Mahadevan. Manifold alignment without correspondence. In *IJCAI*, volume 2, page 3, 2009.

List of Acronyms

MDP	Markov Decision Process
PAC	Probably Approximately Correct
GAN	Generative Adversarial Network
UCB	Upper Confidence Bound